MASTER EN SCIENCES INFORMATIQUES

Outil d'aide à l'enseignement assisté par ordinateur de l'analyse combinatoire

Ganhy, Daniel

Award date:
1984

Awarding institution:
Universite de Namur

Année Académique 1983 - 1984

OUTIL D'AIDE A L'ENSEIGNEMENT ASSISTE PAR ORDINATEUR DE L'ANALYSE COMBINATOIRE

Promoteur : Mr CHERTON

Mémoire présenté par Daniel GANHY
en vue de l'obtention du grade de
licencié et maître en informatique

Je tiens à remercier Monsieur Cherton, promoteur de ce mémoire, qui me prodigua des conseils judicieux à la base de ce travail ainsi que Mademoiselle Rasselle, Mesdames Fauconnier et Higuet, enseignantes en mathématiques, pour les nombreuses heures qu'elles me consacrèrent pendant un an.

Je remercie également Jacqueline pour le soin qu'elle a apporté à l'impression de ce mémoire et enfin ma fiancée Patricia notamment pour les rôles de lectrice et correctrice qu'elle a tenus.

Table des matières

Chapitre d'introduction : objectifs du travail.

0.1. Quel type d'enseignement assisté par ordinateur ?

Nous sommes en 1984. Depuis quatre, voire cinq ans, le cours d'informatique est au programme de certaines écoles secondaires en Belgique. Dans la majorité de ces écoles, cet enseignement utilise le support des micro-ordinateurs. Le prix de ces machines, tout comme celui des ordinateurs plus puissants, ne cesse de diminuer, à un point tel qu'elles sont accessibles à toutes les écoles. Maintenant se pose avec acuité le problème de leur utilisation : les écoles qui ne peuvent compter sur un professeur qui pourrait assurer la gestion du matériel informatique n'en achètent évidemment pas.

Néanmoins, profitant de l'introduction du cours d'informatique et de la baisse de prix du matériel, l'informatique en tant qu'aide à l'enseignement connait un développement inimaginable auparavant. En effet, pour les écoles, le coût constituait un désavantage rédhibitoire pour l'utilisation de l'outil informatique.

Dans ce cadre, l'enseignement assisté par ordinateur est susceptible d'un développement et d'une mutation suffisamment importants pour en modifier tous ses fondements actuels. Il y a encore peu, enseignement assisté par ordinateur signifiait gros systèmes et n'était donc que très peu disponible pour les écoles secondaires, si ce n'est au moyen de lignes louées reliées au centre. Même dans ce cas, seule une école très importante peut se permettre d'utiliser une telle technique qui est malgré tout assez coûteuse. Dans notre pays, citons par exemple le système PLATO, développé à l'université d'Illinois et dont on se sert à l'université Libre de Bruxelles et dans quelques écoles environnantes.

Actuellement, après cette introduction des micro-ordinateurs à l'école, quelques constructeurs ou producteurs

de logiciels ont saisi tout l'intérêt que pourrait représenter un tel marché pour eux : ainsi sont apparus des produits en général assez pauvres du point de vue pédagogique.

Pour pallier aux désavantages respectifs des gros systèmes et des produits "commerciaux", il convient d'appliquer deux remèdes : utiliser les micro-ordinateurs et garantir la valeur pédagogique des futurs produits.

En ce qui concerne le premier remède, nous pensons que, disposant d'un budget donné, les écoles devraient miser plus sur la quantité que sur la puissance même si, dans ce cas, un minimum de contraintes doivent être respectées. Mieux vaut en effet posséder dix petites machines qu'un bel ordinateur, sur lequel il faudrait veiller jalousement et qui serait donc très peu accessible.

La remédiation au second désavantage nécessite plusieurs contraintes. Tout d'abord, les enseignants doivent absolument être impliqués dans l'élaboration des nouveaux projets. Sinon, le risque est grand de produire de beaux outils au point de vue technique mais sans aucune valeur pédagogique.

Ensuite, les enseignants qu'ils aient été impliqués ou non dans l'élaboration du système qu'ils utiliseront, devront pouvoir l'adapter à leurs besoins, leurs désirs propres.

Il convient donc de ne pas construire des programmes d'enseignement assisté par ordinateur de style "boîte noire". Sinon, les enseignants n'adopteront pas ces nouveaux outils et les laisseront de côté.

Enfin, compte tenu du fait que de plus en plus de professeurs ont actuellement reçu une formation minimale en informatique, ces projets devront être compréhensibles par eux. Ils devront donc être rédigés en un langage de haut niveau qui leur soit familier (BASIC, PASCAL,...) et accom-

pagnés de toute la documentation nécessaire. En outre, s'ils sont désireux de réaliser eux-mêmes des programmes d'aide à l'enseignement, ils devront leur servir de modèles de bonne programmation.

0.2. Pourquoi l'analyse combinatoire ?

Ce travail d'enseignement assisté par ordinateur traitera de l'analyse combinatoire. Expliquons ce choix : selon Bloom [1], "si l'on garantit, à tous les élèves, la maîtrise des comportements cognitifs de départ avant d'aborder un nouvel apprentissage, on augmente la qualité moyenne de l'enseignement et on réduit de 50 % la variabilité généralement observée dans les résultats des élèves". On peut raisonnablement penser que les élèves auxquels on enseigne cette matière possèdent les prérequis nécessaires. En effet, il s'agit uniquement de maîtriser suffisamment le calcul. Donc les résultats devraient être assez bons.

Pourtant, beaucoup s'accordent pour reconnaître que les résultats obtenus par les formes traditionnelles d'enseignement sont assez moyens. De plus, les meilleurs résultats ne sont pas toujours le fait des élèves les plus à l'aise dans la majorité des matières. Ceci tend à renforcer l'argument selon lequel l'importance des prérequis est minime lors de l'apprentissage de l'analyse combinatoire. Plusieurs facteurs peuvent expliquer ces résultats : tout d'abord, le peu d'heures que les enseignants peuvent consacrer à cette matière; ensuite, le type de raisonnement qui doit être utilisé est assez inhabituel dans l'ensemble des autres matières enseignées. Il est possible qu'en étant confrontés à des exercices plus nombreux et plus diversifiés, les élèves ne seraient plus amenés par la force des choses à utiliser des formules sans maîtriser parfaitement leur champ d'application. Leur travail serait donc plus profond et irait plus dans le sens d'une compréhension réelle des mécanismes de résolution de problèmes. Donc, en cette matière, un enseignement assisté

par ordinateur pourrait disposer d'un avenir prometteur.

0.3. Présentation de la suite du travail.

Nous allons maintenant présenter le sommaire de la suite du travail en accordant une attention plus particulière au premier chapitre dans le but de définir les idées maîtresses relatives au type d'enseignement assisté par ordinateur employé.

Tout d'abord, suite à nos rencontres avec les enseignants, nous avons plutôt opté vers un enseignement essentiellement composé d'exercices. Dans cette optique, deux formes d'enseignement assisté par ordinateur ont été abordées. D'une part, une forme classique a été envisagée : dans ce cas, pour un exercice, il est nécessaire de mémoriser l'énoncé, la solution, éventuellement les erreurs courantes et les explications à fournir aux élèves dans chacun de ces cas. Tout ceci constitue le scénario, habituellement divisé en pages qui correspondent aux grilles d'écran. Une fois ces pages enregistrées dans l'ordinateur, nous devons organiser leur séquence en fonction des réponses que va donner l'élève. Pour qu'un tel produit soit acceptable, nous devons faire appel à des spécialistes de la discipline, donc des enseignants en mathématique; ainsi qu'à des spécialistes de la technique employée, donc des informaticiens. Cette façon de travailler nous a tout de même paru assez coûteuse : elle exige beaucoup de place en mémoire et beaucoup de travail pour rédiger le scénario.

D'autre part, une forme d'enseignement assisté par ordinateur plus "intelligente" a été étudiée. Ici, il ne s'agit plus d'enregistrer les explications pour chaque exercice, ni donc de les rédiger mais de les générer par programme. Pour cela, à l'énoncé en langage naturel, nous avons décidé d'annexer un autre énoncé en langage formalisé qui permet à l'ordinateur de produire des explications mais aussi de cal-

culer la solution à l'exercice posé. Le système est donc doté de "connaissances" contenues dans un programme qui reçoit en entrée un énoncé en langage formalisé et fournit en sortie la solution et les explications. Ce projet est évidemment plus complexe mais bien plus rentable : en effet, celui-ci une fois terminé, ajouter un exercice requiert un minimum de temps, il s'agit uniquement de rédiger la traduction de l'énoncé en langage formalisé et d'enregistrer les deux types d'énoncé dans la machine. Que l'on compare ceci avec le temps nécessaire pour ajouter un exercice à la forme classique présentée ci-dessus : dans ce cas-là, nous devons identifier les erreurs courantes; pour chacune d'entre elles, il convient de rédiger les explications adéquates et d'introduire le tout dans l'ordinateur. C'est principalement pour cette raison que nous avons choisi de développer la seconde forme d'enseignement assisté par ordinateur.

Le second chapitre consiste en la définition du langage. D'abord, nous tentons de voir à quels objectifs et contraintes ce langage doit satisfaire et tentons de trouver le type de présentation le plus approprié pour les enseignants qui en seront les lecteurs. Cette présentation possèdera un double but : elle servira à la fois de texte de ce travail et de manuel de référence pour les enseignants. En ce qui concerne les langages de programmation, nous trouvons généralement deux textes différents : un manuel utilisateur, destiné à ceux qui apprennent le langage et donc truffé d'exemples et de résumés et un rapport qui constitue une référence concise pour les programmeurs et implémenteurs. Quant à nous, nous avons pensé que le texte de ce mémoire consacré au langage ne devait pas être trop concis sous peine de rendre sa lecture trop ardue. Le langage est en effet assez nouveau que ce soit pour les informaticiens ou pour les enseignants. D'autre part, nous avons cru souhaitable de laisser aux enseignants, futurs utilisateurs, leur propre rapport traitant des

concepts qu'ils désireront utiliser. Ensuite, nous présentons le langage en commençant par le schéma qui est à la base de sa définition suivi des concepts de base introduits par le biais des formules classiques de l'analyse combinatoire telles les arrangements, combinaisons, permutations simples et avec répétition et des extensions à ces concepts de base. La dernière partie de ce chapitre a trait aux formules permettant de calculer la solution à un énoncé d'analyse combinatoire traduit dans le langage.

L'analyse fonctionnelle du système fait l'objet du troisième chapitre. Le système est schématisé dans le but de distinguer les différentes procédures le constituant. Tout d'abord, nous décrivons la procédure qui est chargée de dégager la solution de l'exercice à partir de l'énoncé en langage formalisé et qui la compare à celle fournie par l'élève. Ensuite nous présentons la procédure chargée d'expliquer à partir de l'énoncé en langage formalisé la solution ou une partie de celle-ci qui est plus ou moins grande en fonction de la réponse donnée par l'élève. Nous expliquons pourquoi cette procédure nécessite d'abord une procédure préalable de traduction qui transforme l'énoncé en langage formalisé en une structure de données plus aisément manipulable par la procédure d'explication proprement dite.

Enfin, le quatrième chapitre a trait à l'analyse organique de ces procédures. Elle comprend la structuration hiérarchique du système, la spécification, la programmation et le codage en Pascal des différents modules ainsi que des jeux de test.

Chapitre 1. Approche didactique.

1.1. Présupposé : cours théoriques préalables à l'enseignement assisté par ordinateur.

Après plusieurs contacts avec les enseignants, nous nous sommes rendus compte que ceux-ci ne désirent pas que

l'aide que nous allons apporter constitue la première approche de l'analyse combinatoire. Ils ne craignent pas que leurs prérogatives leur soient enlevées ni que leurs emplois soient menacés mais ils pensent qu'il est préférable que ce soit eux qui introduisent une matière aussi nouvelle. Dans le reste du travail, on supposera donc que les élèves ont déjà suivi des cours traitant de l'analyse combinatoire. Cela signifie donc que le cours que nous préparerons sera essentiellement composé d'exercices et éventuellement accompagné de rappels de théorie. Dans cette hypothèse, plusieurs types d'enseignement assisté par ordinateur sont encore imaginables.

1.2. Forme traditionnelle d'enseignement assisté par ordinateur.

Tout d'abord, une forme qu'on peut qualifier de traditionnelle peut être envisagée. Expliquons-nous [2]: historiquement, Skinner, spécialiste du conditionnement et de l'apprentissage peut sans doute être considéré comme le "père spirituel" de l'enseignement assisté par ordinateur. En effet, il a introduit dans les années 50 l'enseignement programmé qu'il a défini comme "procédé pédagogique qui consiste à présenter le matériel à apprendre sous formes d'étapes successives et bien structurées". Plusieurs techniques permettent d'appliquer ce procédé. D'abord, des livres et des machines ont été construites. Ensuite, grâce à ses possibilités de mémoire et de décision, l'ordinateur a été utilisé et c'est ainsi qu'est né l'enseignement assisté par ordinateur.

En suivant ces préceptes, nous pouvons procéder comme suit : nous développons le scénario en collaboration avec les enseignants; nous définissons ainsi la séquence souhaitée des unités d'apprentissage qui seront présentées aux élèves ainsi que le contenu de celles-ci. Il est souhaitable que ce contenu soit aussi "paramétrisable" que possible : les nombres utilisés doivent pouvoir être variables et les

exercices doivent pouvoir être remplacés par d'autres assez semblables auxquels pourrait s'appliquer la même séquence d'explications. Ainsi chaque élève apprendrait à l'aide d'exercices différents et de même, un élève utilisant plusieurs fois une même partie de cet enseignement découvrirait un contenu différent.

Au point de vue de l'élève, cette méthode d'apprentissage possède plusieurs avantages

. D'abord, elle possède ceux de l'enseignement programmé, à savoir : si le scénario est bien conçu, elle permet une adaptation permanente aux difficultés d'assimilation de l'élève.

. l'élève participe activement à la leçon.

. l'acquis est corrigé immédiatement et point par point.

Ensuite l'utilisation de l'ordinateur offre des possibilités supplémentaires :

. les élèves sont très intéressés par ce moyen pédagogique nouveau qui de plus est actuellement pour eux synonyme de jeux.

. ses possibilités techniques sont attrayantes : suivant les machines utilisées (nombre de points définis, nombre de couleurs,...), les graphiques peuvent être plus ou moins développés; de plus ces micro-ordinateurs peuvent souvent émettre des sons appréciés par les plus jeunes.

Cependant, nous pourrions profiter davantage des capacités de l'ordinateur : de cette façon, nous nous servons en effet d'une machine produisant des pages de scénario attrayantes en fonction de la réponse donnée et non d'un outil capable d'une certaine "intelligence". C'est pourquoi nous avons envisagé une forme d'enseignement assisté par ordinateur plus ambitieuse : doter la machine de "connaissances" nécessaires à enseigner l'analyse combinatoire, profitant du fait que cette matière est limitée et ne nécessite que des réponses précises. Ces deux hypothèses sont en effet assez fondamenta-

les : face à une branche vaste, il est exclu qu'on puisse, dans un petit système (micro-ordinateur), introduire un ensemble de données permettant une connaissance suffisante; de même, la précision des réponses implique que le programme qui les analysera est plus ou moins simple et donc possible pour notre petit système.

1.3. Enseignement assisté par ordinateur "intelligent".

L'idéal serait évidemment que, sur base de l'exercice posé à l'élève, qui doit de toutes façons être mémorisé, le programme réalisé soit capable d'en calculer la solution, de l'expliquer et d'évaluer si celle qui est donnée par l'élève y correspond. Malheureusement, les recherches menées en intelligence artificielle nous ont appris qu'il était utopique de vouloir faire comprendre par un ordinateur un texte en langage naturel.

L'énoncé pourrait être simplifié de façon à être compréhensible à la fois par l'élève et la machine. Pour être accessible à l'ordinateur, on utiliserait donc un vocabulaire restreint et une syntaxe suffisamment limitée, c'est-à-dire ce que l'on nomme en jargon d'informaticien un langage, tout en restant assez intelligible pour que l'élève comprenne l'énoncé qui lui est présenté. Sur base de cet énoncé traduit dans ce langage formalisé, la solution devrait donc être calculée et éventuellement expliquée à l'élève si la réponse qu'il fournit est erronnée. Ce procédé présente deux désavantages importants : d'abord, son caractère réalisable n'est pas assuré. En effet, imaginer un tel formalisme n'est pas aisé et son efficacité n'est pas du tout garantie. Ensuite, l'élève ne résoudrait plus un exercice d'analyse combinatoire mais un exercice "traduit" ou "standardisé". Or, les enseignants que nous avons rencontrés nous ont signalé que comprendre l'exercice posé représente souvent une difficulté capitale qui dans ce cas-ci serait esquivée.

Pour pallier à ce désavantage, nous avons pensé mémoriser deux énoncés : l'un en langage naturel présenté à l'élève et l'autre en langage formalisé. Le premier n'aurait comme but que d'être présenté tel quel à l'élève, le second permettrait à l'ordinateur de calculer la réponse et de l'expliquer à l'élève. Le caractère réalisable de cette solution n'est pas assuré non plus mais il est tout de même plus problable que celui du procédé précédent. En effet, l'énoncé traduit en langage formalisé ne doit pas être compris, ni même vu par l'élève. Une plus grande complexité peut donc être admise. Il n'en reste pas moins qu'un ensemble suffisamment vaste d'exercices d'analyse combinatoire doivent pouvoir être exprimés dans ce formalisme. De plus, à l'aide de celui-ci, le système doit être capable de calculer la réponse de manière non équivoque. En supposant que ce projet soit réalisable, nous allons tenter de le décrire et de l'analyser plus précisément pour pouvoir comparer ses avantages et désavantages à ceux de la première forme d'enseignement assisté par ordinateur décrite en 1.2. et ce au vu des objectifs définis dans la première partie de l'introduction.

Dans ce cas, l'enseignement consiste donc en la présentation, la correction et l'explication d'exercices mémorisés. Pour chacun d'entre eux, deux énoncés doivent être enregistrés, l'un en langage naturel, l'autre en un langage formalisé restant à définir. Les solutions et leurs explications ne doivent en effet pas être mémorisées pour chaque exercice dans ce cas-ci puisqu'elles sont générées par programme à partir des énoncés formalisés. Ces explications sont donc nécessairement indépendantes des exercices précédemment présentés aux élèves. Par conséquent, nous n'avons pas intérêt à définir une séquence d'exercices de façon définitive puisque les explications ne pourront faire appel à ce qui a déjà été vu. Au contraire, il est intéressant que les enseignants définissent eux-mêmes pour chaque élève la séquence d'apprentissage ou encore que les élèves la définissent eux-mêmes. Elle devra aussi pouvoir se déci-

der dynamiquement, c'est-à-dire en fonction des réponses fournies aux exercices faits précédemment. Nous devrons donc disposer d'un programme permettant de mémoriser cette chronologie.

Cette utilisation d'exercices enregistrés et donc rédigés par d'autres, qu'ils soient concepteurs du système ou enseignants peut ne pas être satisfaisante pour le professeur concerné. A un stade ultérieur, il peut désirer introduire d'autres exercices que ceux déjà mémorisés; rappelons que pour enregistrer des exercices dans la machine, il faut introduire non seulement l'énoncé en langage naturel, ce qui est évident à réaliser, mais aussi sa traduction dans le langage formalisé que l'enseignant devra donc apprendre. Il disposera ainsi d'exercices supplémentaires. Ensuite, en tenant compte de ces ajoûts, il pourra redéfinir la séquence d'apprentissage.

De plus, des élèves suffisamment aguerris, pourraient eux aussi apprendre le langage formalisé pour, à leur tour, rédiger des exercices que la machine analyserait. Comme elle le fait pour l'énoncé qui est introduit par le professeur, elle "saurait" calculer la solution de ce second énoncé traduit en langage formalisé et comparer ces deux solutions. Si elles ne sont pas équivalentes, nous pouvons être certains que l'élève n'a pas bien compris l'énoncé (à condition bien sûr qu'il maîtrise le langage) et donc, dans ce cas, lui demander une réponse n'est pas utile. Par contre si elles sont équivalentes, l'élève est supposé avoir bien "décodé" l'exercice (à moins d'un effet du hasard). Il serait donc invité à fournir une réponse à l'énoncé posé et, en cas d'erreur, nous pouvons penser qu'il s'agit plus d'une erreur de calcul que d'une erreur de raisonnement et nous pouvons fournir les explications adéquates. Notons que même dans le premier cas (solutions équivalentes), il n'est pas impossible que quelquefois l'analyse de l'énoncé formalisé introduit par l'élève comparée à celle fournie par l'enseignant nous permet-

te de lui signaler où il a fait une erreur dans sa compréhension de l'énoncé. Or c'est précisément à ce moment-là que se posent beaucoup de problèmes : comprendre l'énoncé représente souvent une difficulté capitale pour les élèves (cf p.9.ci-dessus), de là tout l'intérêt d'une telle utilisation de cette forme d'enseignement assisté par ordinateur.

Quant au professeur, il jouerait un rôle que nous jugeons très intéressant : il se concentrerait davantage sur l'explication du sens de l'énoncé que sur l'explication des détails de calculs qui serait du ressort de la machine.

1.4. Etude des deux formes d'enseignement assisté par ordinateur au point de vue du professeur sachant programmer.

Comme décrit p. 2/3, un objectif de ce projet est d'être compréhensible et adaptable par des professeurs ayant reçu une formation en informatique. Voyons ce que signifie une approche des deux formes d'enseignement assisté par ordinateur dont le centre d'intérêt est cette fois le professeur. Dans le premier cas, excepté pour l'analyse des réponses qui risque d'être plus ardue, des connaissances approfondies en informatique ne sont pas requises pour la première forme d'enseignement (cf 1.2). En effet, il s'agit principalement d'afficher à l'écran des pages de scénario; les techniques employées ne sont pas particulières à la matière choisie. Ce projet est donc aisément adaptable et peut de plus, le cas échéant, servir de modèle à l'enseignant désireux de construire ses propres leçons.

Pour la seconde forme d'enseignement (cf 1.3), le problème est assez différent : les techniques sont plus complexes. Cependant, en structurant le système d'une façon adéquate, la procédure d'explication peut être simplifiée par l'utilisation de modules de bas niveau qui eux seraient plus complexes. Elle peut donc être rendue compréhensible par des enseignants n'ayant pas reçu une formation complète en infor-

matique. Ils pourront adapter cette procédure de haut niveau tout en utilisant les outils plus complexes mis à leur disposition.

1.5. Résumé des différences entre les deux formes d'enseignement assisté par ordinateur.

Avant de comparer les avantages et désavantages respectifs des deux formes d'enseignement assisté par ordinateur envisagées jusqu'ici, résumons leurs différences.

Examinons d'abord ce qui doit être mémorisé dans chacun des cas : il s'agit, d'un côté, de l'énoncé, des différentes solutions; de l'autre, il s'agit de deux énoncés, l'un en langage naturel bien sûr et un autre en langage formalisé.

Ensuite, dans le premier cas, le programme est assez simple à réaliser : il consiste essentiellement en l'affichage des pages de scénario; dans l'autre cas, il est plus comlexe : il doit permettre de déduire, à partir de l'énoncé en langage formalisé, la solution ainsi que des explications.

Comparons aussi l'"adaptabilité" de ces différents projets d'enseignement assisté par ordinateur : dans le premier, le professeur doit être initié à la programmation pour adapter le programme à ses désirs. Dans cette éventualité, il pourra aussi prendre le programme réalisé comme modèle pour d'autres leçons qu'il désirerait développer. En ce qui concerne le second projet, si l'enseignant ne désire y investir aucun travail, il peut l'utiliser comme une boîte noire contenant les exercices que les concepteurs du système ou d'autres enseignants ont mémorisé à sa place. Par contre, s'il fait l'effort d'apprendre le langage formalisé (ce qui est tout de même le but de cette forme d'enseignement assisté par ordinateur), il introduit lui-même les exercices qu'il désire voir figurer dans le programme d'apprentissage de ses élèves. Si de plus, l'enseignant sait programmer, il peut

modifier le programme générant les explications à condition que celui-ci soit bien structuré et bien commenté. Donc, plus l'enseignant sera qualifié, plus l'enseignement assisté par ordinateur correspondra à ce qu'il en attend.

Enfin, rappelons que la seconde forme d'enseignement possède une dimension supplémentaire par rapport à la première. En effet, en rédigeant des exercices dans le langage formalisé, les élèves apprennent à exprimer de façon rigoureuse ce que signifie l'énoncé en langage naturel.

1.6. Choix de la solution à développer.

Il nous reste donc à comparer les deux formes d'enseignement assisté par ordinateur vues en 1.2 et 1.3. Supposons que la seconde forme soit réalisable : si elle devait être choisie, on essayerait d'imaginer un formalisme et en cas d'échec, on développerait évidement le premier projet.

Rappelons les avantages de cette première forme : tout d'abord, elle possède les avantages de l'enseignement programmé; ensuite, l'utilisation de l'ordinateur est attrayante pour les élèves; enfin, les techniques employées en programmation peuvent être simples et le programme est donc susceptible de servir de modèle aux enseignants. Elle a comme désavantage principal de nécessiter beaucoup de mise en oeuvre : en effet, rédiger des explications pour chaque type de solution à chaque exercice nécessite beaucoup de temps et les stocker exige beaucoup de mémoire. De plus, ce type d'enseignement est assez difficile à étendre si ce n'est pour les enseignants initiés à la programmation. Cette non-extensibilité peut avoir pour conséquence que les enseignants se sentent peu concernés par ce produit à la réalisation duquel tous n'ont pas pu participé. Or (cf 0.1), ceci est une cause importante de non utilisation de ce type d'outils informatiques.

La seconde forme possède, elle aussi, les avantages de l'enseignement programmé : puisque c'est le professeur qui définit pour chaque élève la séquence d'apprentissage, elle possède les meilleures chances d'être adaptée aux difficultés d'assimilation de l'élève. En ce qui concerne la participation active de l'élève et la correction immédiate de son acquis, aucune différence avec l'autre forme n'est possible. De plus, l'utilisation de l'ordinateur n'a aucune raison d'être moins attrayante. Par contre, les techniques employées sont plus complexes mais peuvent être uniquement concentrées dans des modules de bas niveau. Le programme générant peut en effet être rendu simple en utilisant des modules de bas niveau utilitaires qui doivent être spécifiés de la façon la plus précise possible. Enfin cette forme a comme avantage que le professeur peut mieux diriger l'ordre dans lequel les exercices seront présentés aux élèves puisqu'il peut définir la séquence d'apprentissage. Ceci a comme désavantage que l'on ne peut inclure dans les explications des références à des exercices déjà présentés et la juxtaposition des exercices risque donc de paraître assez disparate à l'élève même si celle-ci a été longtemps réfléchie. Un autre désavantage est possible : un même programme devant générer des explications pour tous les exercices, celles-ci risquent d'être rendues moins explicites que celles du premier type qui sont particulières à chaque exercice. A nous d'employer un vocabulaire précis et significatif. Sans doute le fait d'utiliser des graphiques est-il susceptible de rendre ces explications plus compréhensibles.

Si les enseignants désirent utiliser l'enseignement assisté par ordinateur comme une boîte noire, le choix est assez difficile à effectuer.

Cependant, dans l'introduction (cf 0.1), nous avons rejeté cette hypothèse, source de non-utilisation de ce type d'outils informatiques. En ce qui concerne la seconde forme, nous proposons donc que les enseignants fassent l'effort d'apprendre le formalisme. Nous pensons que leur effort sera

largement récompensé par l'ajoût d'exercices qui leur sont personnels. En effet, les enseignants nous ont appris que chacun, ils utilisent des exercices différents qu'ils estiment représentatifs de la matière.

Sous ces conditions, la seconde forme paraît bien plus avantageuse : en effet, les enseignants pouvant introduire eux-mêmes des exercices, le système connaîtra une vie très dynamique. De plus, cet ajoût d'exercices requiert assez peu de temps : le rapport du nombre d'heures d'enseignement produites sur le nombre d'heures de travail est très avantageux. N'oublions pas non plus que l'élève a la possibilité d'introduire des exercices, ce qui nous paraît pédagogiquement assez intéressant. Enfin, dans le cadre d'un projet de recherche, il est souhaitable de réaliser le travail le plus original possible dans une voie qui est susceptible de connaître de nombreux développements.

Donc, si ce projet est réalisable, nous développerons la seconde forme. Notons pour terminer que l'on peut combiner ces deux formes : l'enseignement peut disposer de pages de texte consistant en des rappels de théorie qu'il insérera dans la séquence d'apprentissage de la seconde forme.

Chapitre 2. Définition du langage.

2.1. Objectifs et contraintes du langage.

L'objectif de ce chapitre est donc de définir le formalisme nécessaire pour exprimer les problèmes d'analyse combinatoire. C'est-à-dire qu'il faut convenir de tous les signes et symboles qui seront utilisés par l'enseignant pour décrire l'exercice à l'ordinateur. En jargon d'informaticien, un tel formalisme est appelé langage. Il dictera donc quelles expressions sont admissibles pour la machine. Les caractères disponibles sur un terminal influenceront par conséquent la syntaxe employée.

Idéalement, le langage devrait permettre d'exprimer tous les problèmes d'analyse combinatoire, c'est-à-dire, comme le définit [3], les problèmes mettant en oeuvre "quelques techniques permettant de déterminer sans dénombrement direct le nombre de résultats possibles d'une expérience particulière". Un tel résultat risque d'être difficile à atteindre; aussi, pour juger si celui que nous obtenons est suffisant dans le cas qui nous intéresse, nous examinerons si nous pouvons traduire les problèmes d'analyse combinatoire que les professeurs de l'enseignement secondaire posent à leurs élèves. A l'aide de ce langage, l'ordinateur doit pouvoir calculer la solution et l'expliquer à l'élève. Il est donc nécessaire d'y inclure des liens avec l'énoncé en langage naturel pour que l'élève ne soit pas désemparé devant des explications trop abstraites, sans rapport avec l'énoncé qui lui a été présenté.

Pour réussir dans cette entreprise, nous ne pouvons nous baser uniquememt sur les formules classiques d'arrangements, de permutations et de combinaisons, simples et avec répétition. En effet, si elles nous sont quelquefois très utiles, elles ne suffisent pas à rendre compte d'un ensemble suffisamment vaste de problèmes. Le but à atteindre est

d'"apprendre" à l'ordinateur notre façon de résoudre un exercice pour qu'à son tour, il "puisse l'expliquer". Pour dégager un tel langage, nous allons donc démonter le raisonnement que nous tenons.

Deux personnes différentes utilisent le système : l'enseignant qui décrit les problèmes au système et l'élève qui, grâce au travail de son professeur qui permet de générer les explications, apprend à résoudre ces problèmes. Lors de l'utilisation du système par l'enseignant, deux grands types de discours sont à distinguer : le monologue (l'utilisateur décrit ce qu'il désire sans injonctions ou questions de l'ordinateur. Il s'exprime à l'aide d'un sous-ensemble limité de l'éventail des expressions familières décrivant le problème) et le dialogue qui se subdivise encore en deux formes distinctes : les dialogues de style "questions/réponses" et les dialogues de style "menu". Dans les dialogues de style "questions/réponses", le système pose une question, éventuellement à choix multiple, l'utilisateur (ici l'enseignant) y répond et ainsi de suite jusqu'à ce que le système ait cerné sans ambiguités ce qu'il doit faire. La question posée est fonction de la réponse à la question précédente. Par contre, dans le style "menu", dans une même grille d'écran, toute une série de questions sont posées sans lien de cause à effet entre elles. Cette distinction n'est néanmoins pas tellemnent importante et pas toujours explicite. Dans le cas qui nous intéresse, le style "questions/réponses" peut être combiné avec le style "menu" suivant les cas : au début, quelques questions doivent être inévitablement posées (le style "menu" peut donc être utilisé); ensuite, en fonction des réponses à ces questions indispensables, des questions plus particulières seront présentées (le style "questions/réponses" peut donc aussi être employé).

Comparons ces deux types de discours pour décider lequel choisir pour notre système d'enseignement assisté par ordinateur. En ce qui concerne le dialogue, l'utilisateur

n'a besoin que de très peu d'apprentissage pour utiliser le système : il peut déduire de la question posée à l'écran quelle réponse le système attend de lui. Si le problème auquel il doit faire face est assez complexe, il devra donc répondre à beaucoup de questions et ces intermèdes risquent de le lasser après quelques utilisations du système. Ce désavantage n'existe pas pour le monologue, où, l'utilisateur peut décrire plus précisément ce qu'il veut sans devoir répondre à des questions qui ne sont pas toujours pertinentes dans tous les cas qui nous intéressent à un moment donné. Par contre, l'apprentissage du monologue est plus long et plus difficile : la syntaxe est en effet beaucoup plus complexe. Dans la matière qui nous intéresse, nous préférerons tout de même nous orienter vers le monologue. En effet, il convient mieux pour décrire un raisonnement. Le dialogue est trop statique et deviendrait trop lassant vu le nombre des questions posées et la moindre pertinence de celles-ci.

La description du langage qui va suivre est principalement destinée aux professeurs qui enseignent l'analyse combinatoire. Le lecteur ne doit pas nécessairement posséder une base informatique mais il doit disposer d'une bonne connaissance de l'analyse combinatoire, l'objectif étant que les professeurs rédigent eux-mêmes les exercices dans ce langage.

Après plusieurs tentatives, il est apparu souhaitable de faire autant que possible le lien avec des notions familières aux enseignants telles les arrangements, permutations et combinaisons simples et avec répétition. Sans ces liens avec la théorie classique, les professeurs se trouvaient désemparés face à des concepts aussi nouveaux et ne percevaient pas toujours le rapport avec ce qu'ils connaissaient bien. C'est pourquoi nous allons montrer aussi tôt que possible comment des énoncés simples faisant appel à une seule de ces notions à la fois peuvent être traduits dans le langage.

2.2. Présentation du langage.

2.2.1. Schéma de base.

Pour disséquer notre façon de raisonner, commençons par résoudre un exercice simple [4 ex 3-5 1° p 11] en oubliant toutes les formules classiques d'analyse combinatoire.

Exercice n° 1.

- Combien y a-t-il de nombres écrits avec 2 chiffres qui soient différents et distincts de 0 ?

- . nombre de chiffres $\neq 0$: 9
  . nombre de chiffres dans le nombre écrit : 2
  donc, 9 possibilités pour le 1er chiffre
  8 possibilités pour le 2eme. (les 9 chiffres possibles - celui utilisé pour le 1er chiffre)
  au total, il y a donc 72 nombres.

1er chiffre du nombre — 2eme chiffre du nombre

9	X	8	=	72

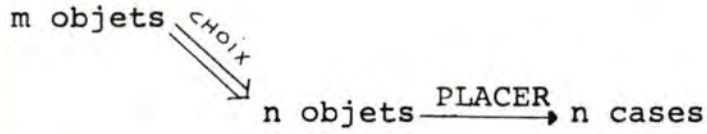
En examinant le raisonnement tenu, voici comment on a procédé (le vocabulaire utilisé est celui de [3])

1. Combien d'objets sont-ils disponibles ? (dans ce cas, les objets correspondent aux chiffres $\neq 0$)

2. Combien en choisit-on pour les placer dans des cases ? (dans ce cas, les cases correspondent aux chiffres du nombre)

N.B. D'autres sources, comme par exemple [5] utilisent plutôt le terme "groupement". Donc les cases (vocable utilisé par [3]) correspondent aux éléments du groupement (vocable utilisé par [5])

On peut schématiser le raisonnement utilisé comme suit :



$n \leq m$

Résoudre un exercice de ce type consiste à dénombrer le nombre de façons de placer n objets choisis parmi m dans n cases (1 case contenant 1 et 1 seul objet)

Passons à un autre exercice toujours aussi simple.

Exercice n° 2.

- Combien y a-t-il de plaques minéralogiques constituées uniquement de 2 caractères, parmi lesquels la lettre A obligatoirement et un chiffre au choix ?

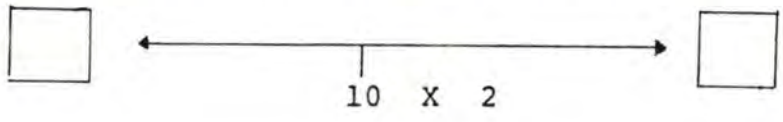
- 2 ensembles d'objets : les chiffres (au nombre de 10)
  la lettre A (au nombre de 1)

pour le premier ensemble :

. on choisit un élément parmi les 10
. pour le placer, on a le choix entre 2 cases (les 2 caractères de la plaque minéralogique)

la lettre A se place dans la case non choisie ci-dessus.
au total, il y a donc 20 plaques minéralogiques.

ler caractère de la plaque 2eme caractère de la plaque



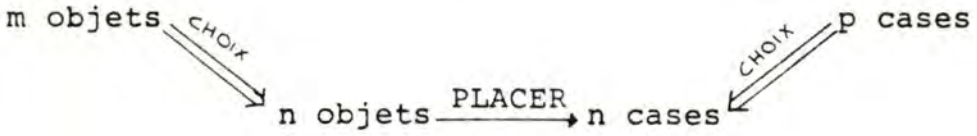
Le modèle de raisonnement décrit ci-dessus peut être étendu de la manière suivante :

1. Combien d'objets sont-ils disponibles ? (dans ce cas, les objets correspondent aux chiffres)

2. Combien de cases sont-elles disponibles ? (dans ce cas, les cases correspondent aux caractères de la plaque)

3. Combien choisit-on d'objets pour les placer dans les cases choisies ?

On obtient ainsi le schéma de base du langage :



$n \leqslant m$ $\qquad\qquad$ $n \leqslant p$

Résoudre un exercice consiste maintenant à dénombrer le nombre de façons de placer n objets choisis parmi m dans n cases choisies parmi p (1 case choisie contenant 1 et 1 seul objet).

Il reste à traduire ce raisonnement par un monologue en faisant le lien avec l'énoncé de l'exercice pour respecter ce qui a été décidé en 2.1.

Un exercice consistera en deux parties essentielles : la description des actions à effectuer pour résoudre l'exercice et la description des ensembles manipulés par ces actions qu'on appellera déclarations. Chaque action correspond à une étape de résolution de l'exercice et une formule y sera associée pour nous permettre de calculer la solution de l'exercice.

Traduction du schéma de base :

(nom de l'ensemble d'objets) # m;
(nom de l'ensemble de cases) # p;
placer n de (nom de l'ensemble d'objets) dans n de (nom de l'ensemble de cases).

Donner les noms des ensembles d'objets et de cases nous permet de faire le lien avec l'énoncé. La mise entre parenthèses signale que cette partie va être remplacée dans les cas particuliers par ce qui est décrit à l'intérieur des parenthèses. Le signe a son interprétation mathématique habituelle : le nombre qui suit est donc le nombre d'éléments que comprend l'ensemble. ";" et "." sont des séparateurs : ";" sépare deux lignes d'exercices et "." sépare deux exercices.
L'action est la traduction la plus conforme possible du schéma de base :
placer dans exprime la flèche placer du schéma.
de exprime la flèche de choix.

Les mots soulignés sont des mots réservés et ne peuvent être utilisés pour nommer les différents ensembles.

2.2.2. Traduction des formules classiques de l'analyse combinatoire.

Après avoir traduit le schéma de base, nous allons voir comment traduire les différentes formules classiques de l'analyse combinatoire. Chacune d'entre elles sera étudiée en la comparant avec celles vues auparavant, ce qui nous permettra de voir quel concept nouveau doit être introduit dans le langage. Ensuite la traduction de cette formule théorique et d'un exercice s'y rapportant sera donnée.

1. Arrangements simples.

. Définition [5] : On appelle arrangements simples de m objets distincts pris n à n (n entier positif $\leqslant$ m) les divers groupements que l'on peut former en prenant chaque fois n de ces objets, deux groupements distincts devant différer soit par l'ordre de présentation de ces objets, soit par la nature d'un objet au moins.

. Voyons maintenant comment faire correspondre cela à ce que nous avons vu :
- le nombre d'éléments d'un groupement, c'est pour nous le nombre de cases choisies ou encore le nombre d'objets choisis puisqu'une case choisie contient 1 et 1 seul objet choisi.
- deux groupements distincts devant différer par l'ordre de présentation de ces objets signifie que l'ordre dans lequel se trouvent les cases n'est pas indifférent.
- deux groupements distincts devant différer par la nature d'un objet signifie que l'identité de l'objet choisi a de l'importance.

Traduction

(nom de l'ensemble d'objets) # m;
(nom de l'ensemble de cases) # n;
placer n de (nom de l'ensemble d'objets) dans n de (nom de l'ensemble de cases).

exercice n° 1 (cf p.20)

CHIFFRES-DISTINCTS-DE-0 # 9;

CHIFFRES-DU-NOMBRE # 2;

PLACER 2 DE CHIFFRES-DISTINCTS-DE-0 DANS 2 DE CHIFFRES-DU-NOMBRE.

2. Permutations simples.

. **Définition [5] : On appelle permutations simples de n objets distincts les divers groupements que l'on peut former en prenant chaque fois tous les objets, deux groupements distincts devant différer par l'ordre de présentation des objets.**

. **C'est le même type d'exercice que les arrangements sinon qu'ici, tous les objets sont choisis.**

Traduction

> (nom de l'ensemble d'objets) # n;
> (nom de l'ensemble de cases) # n;
> placer n de (nom de l'ensemble d'objets) dans n de (nom de l'ensemble de cases).

exercice n° 3 [3 n° 2.5 (i) p 25] : De combien de façons différentes peut-on répartir un groupe de 7 personnes sur une rangée de 7 chaises ?

```
PERSONNES # 7;
CHAISES # 7;
PLACER 7 DE PERSONNES DANS 7 DE CHAISES.
```

3. Combinaisons simples.

. **Définition [5] : On appelle combinaisons simples de m objets distincts pris n à n (n entier positif $\leq$ m) les divers groupements que l'on peut former en prenant chaque fois n de ces objets, deux groupements distincts devant différer par la nature d'un objet au moins.**

. **A la différence des arrangements simples, deux groupements différant par l'ordre de présentation des objets ne sont pas distincts. Cela signifie donc que les cases sont indis-**

tingables.

Voyons cette différence sur deux exemples.

exercice n° 4

Une assemblée de 12 personnes désire élire en son sein un comité de direction composé d'un président, d'un secrétaire et d'un trésorier.
De combien de façons peut-elle le faire ?

Il y a 12 objets (personnes)
3 cases (membres du comité de direction)
On choisit 3 objets pour les placer dans les cases.

président		secrétaire		trésorier	
12	X	11	X	10	= 1320

exercice n° 5

Une assemblée de 12 personnes désire élire en son sein un comité de direction composé de 3 membres. De combien de façons peut-elle le faire ?

Ici aussi, il y a 12 objets (personnes)
3 cases (membres du comité de direction)
On choisit 3 objets pour les placer dans les cases.
Mais

membre		membre		membre	
12	X	11	X	10	$= \frac{1320}{6} = 220$

$$\frac{}{3\,!\ (6)}$$

Ici les cases sont indistingables. Supposons que l'on ait

choisi les 3 personnes A, B et C. Dans l'exercice n° 4,

président	secrétaire	trésorier	
A	B	C	
A	C	B	Cela correspond à 6
B	A	C	façons possibles
B	C	A	
C	A	B	
C	B	A	

Par contre, dans l'exercice n° 5, peu importe la personne qui est dans la première case (donc qui est 1er membre), celle qui est dans la deuxième, etc...

La différence entre les arrangements simples et les combinaisons simples consiste en un changement de nature des cases (les cases sont indistingables) qui doit être acté dans la description de l'exercice pour que la machine puisse faire la différence lors des calculs et explications. Puisque les ensembles sont décrits dans les déclarations, c'est dans celles-ci que nous avons choisi de mentionner ce changement de nature.. Nous devrons donc dorénavant préciser dans les déclarations si les cases sont distingables ou non.

Par convention, . lorsque les cases seront distingables, on n'ajoutera rien aux déclarations actuelles. Ce sera donc l'option par défaut, celle que le système choisira à défaut d'autres précisions.

. lorsque les cases seront indistingables, après le nombre de cases, on placera le symbole "!" qui fait penser à la division par la factorielle d'un nombre.

. à la différence des arrangements simples, un objet peut se répéter dans un groupement. Ceci a pour conséquence qu'un groupement peut comprendre plus d'éléments que n'en contient l'ensemble d'objets, (c'est-à-dire n pas nécessairement $\leq$ m). Cela signifie qu'un même objet peut être placé dans plusieurs cases et que donc, il peut y avoir plus de cases que d'objets.

Traduction

> (nom de l'ensemble d'objets) # m;
> (nom de l'ensemble de cases) # n!;
> placer n de (nom de l'ensemble d'objets) dans n de (nom de l'ensemble de cases).

traduction des 2 exercices ci-dessus n° 4 et 5

```
PERSONNES # 12;
MEMBRES-DU-COMITE-DE-DIRECTION # 3;
PLACER 3 DE PERSONNES DANS 3 DE MEMBRES-DU-COMITE-DE-DIRECTION.
```

```
PERSONNES # 12;
MEMBRES-DU-COMITE-DE-DIRECTION # 3';
PLACER 3 DE PERSONNES DANS 3 DE MEMBRES-DU-COMITE-DE-DIRECTION.
```

4. Arrangements avec répétition.

- Définition [5] : On appelle arrangements avec répétition de m objets distincts pris n à n (n entier > 0) tous les groupements de n objets que l'on peut former, un même objet pouvant se répéter jusque n fois dans un même groupement et deux groupements distincts devant différer soit par l'ordre de présentation des objets, soit par la nature d'un objet au moins.
- A la différence des arrangements simples, un objet peut se répéter dans un groupement. Ceci a pour conséquence qu'un groupement peut comprendre plus d'éléments que n'en contient l'ensemble d'objets, (c'est-à-dire n pas nécessairement $\leq$ m). Cela signifie qu'un même objet peut être placé dans plusieurs cases et que donc, il peut y avoir plus de cases que d'objets.

Examinons cette différence sur des exemples.

a. l'exercice n° 1 (p 20)

Exercice n° 6

b. Combien y a-t-il de nombres écrits avec 2 chiffres distincts de 0 ?

nombre de chiffres ≠ 0 : 9
nombre de chiffres dans le nombre écrit : 2
donc 9 possibilités pour le 1er chiffre.
9 possibilités pour le 2eme (le 1er chiffre peut être répété)
au total, il y a donc 81 nombres.

1er chiffre du nombre		2eme chiffre du nombre		
9	X	9	=	81

La différence entre les arrangements simples et les arrangements avec répétition consiste en un changement de nature des objets (ils peuvent être répétés) qui doit être acté dans la description de l'exercice pour que la machine puisse faire la différence lors des calculs et explications. Puisque les exemples sont décrits dans les déclarations, c'est dans celles-ci que nous avons choisi de mentionner ce changement de nature. Nous devrons donc dorénavant préciser dans ces déclarations si les objets peuvent être répétés ou non.

Par convention, . lorsque les objets ne peuvent être répétés, on n'ajoutera rien aux déclarations actuelles : ce sera l'option par défaut.

. lorsque les objets peuvent être répétés, après le nombre d'objets, on placera le symbole "*".

Traduction

```
(nom de l'ensemble d'objets) # m*;
(nom de l'ensemble de cases) # n;
placer n de (nom de l'ensemble d'objets) dans n de (nom
de l'ensemble de cases).
```

Traduction de l'exercice n° 6

```
CHIFFRES-DISTINCTS-DE-0 # 9*;
CHIFFRES-DU-NOMBRE # 2;
PLACER 2 DE CHIFFRES-DISTINCTS-DE-0 DANS 2 DE CHIFFRES-DU-NOMBRE.
```

5. Combinaisons avec répétition.

- Définition [5] : On appelle combinaisons avec répétition de m objets distincts pris n à n (n entier $>$ 0), tous les groupements que l'on peut former en prenant chaque fois n objets, un même objet pouvant se répéter jusque n fois dans un même groupement et deux groupements distincts devant différer par la nature d'un objet au moins.

- La différence entre combinaisons avec répétition et combinaisons simples est identique à celle entre arrangements avec répétition et arrangements simples : les objets peuvent donc être répétés.

Traduction

```
(nom de l'ensemble d'objets) # m*;
(nom de l'ensemble de cases) # n!;
placer n de (nom de l'ensemble d'objets) dans n de (nom
de l'ensemble de cases).
```

Nous allons maintenant traduire l'exercice suivant :

Exercice n° 7

- Quel est le nombre de scores différents que peuvent marquer 3 dés indistingables ? (score est à prendre au sens de l'ensemble des 3 valeurs et non au sens de somme de ces valeurs)

. Les objets sont les valeurs d'un dé, au nombre de 6 (de 1 à 6). Ils peuvent être répétés. En effet, l'énoncé ne nous dit pas que les valeurs des dés doivent être différentes.

. Les cases sont les dés. Elles reçoivent un objet, c'est-à-dire une valeur, sont au nombre de 3 et sont indistingables.

```
VALEURS-D'UN-DE # 6*;
DES # 3!;
PLACER 3 DE VALEURS-D'UN-DE DANS 3 DE DES.
```

6. Permutations avec répétition.

. Définition [5] : On appelle permutations avec répétition de n objets, parmi lesquels n_1 objets identiques a_1, n_2 objets identiques a_2 ($a_2 \neq a_1$),..., n_p objets identiques a_p ($a_p \neq a_i$ pour tout $i < p$) ($n = \sum_{i=1}^{p} n_i$), tous les groupements que l'on peut former en prenant chaque fois les n objets, deux groupements distincts devant différer par l'ordre de présentation des objets.

. La différence avec les permutations simples réside dans le fait qu'ici, les n objets ne sont pas distincts mais il y a une série d'objets identiques.

Exercice n° 8

Voyons cette différence sur 2 exemples.

- Chaque signal consistant de 6 pavillons alignés, combien

de signaux différents peut-on former à l'aide d'un pavillon rouge, d'un pavillon vert, d'un pavillon bleu, d'un pavillon blanc, d'un pavillon noir et d'un pavillon jaune ?

Il y a 6 objets différents (pavillons). Ils ne peuvent être répétés.

6 cases (places d'un pavillon). Leur ordre a de l'importance : si on le change, on obtient des signaux différents.

6	5	4	3	2	1	720

Exercice n° 9

- Chaque signal consistant de 6 pavillons alignés, combien de signaux différents peut-on former à l'aide de 2 pavillons rouges, 2 pavillons verts, 1 pavillon bleu et un pavillon blanc ?

. Suivons la démarche de résolution de [3 p. 18.]
Si les pavillons rouges et verts étaient distincts entre eux, il y aurait aussi 720 signaux différents.

Mais étant identiques, on peut les permuter entre eux sans changer le signal. Il y a donc $\frac{720}{2!2!}$ signaux différents, c'est-à-dire 180.

A la différence des objets utilisés dans les permutations simples, les objets utilisés dans les permutations avec répétition ne sont plus tous différents. Pour que la machine puisse effectuer le calcul de la solution et l'expliquer, nous devons préciser combien de classes ou de sous-ensembles d'objets identiques (dans la définition ci-dessus c'est p) composent l'ensemble principal d'objets et le nombre d'éléments de chacune de ces classes (dans la définition ci-dessus il s'agit de $n_1, \ldots, n_p$).

Jusqu'à présent, nous avons introduit une distinc-

tion entre les objets appartenant à un ensemble : ils peuvent tous être répétés ou non. De la même manière, nous allons introduire une seconde distinction : les objets d'un ensemble sont tous identiques ou ils sont tous différents.

Cette seconde distinction ne suffit pas à rendre compte des exercices de permutations avec répétition. En effet, l'ensemble principal est composé d'une série de sous-ensembles d'objets identiques et non pas d'objets tous identiques. Pour résoudre ce problème, nous allons introduire un concept supplémentaire : les ensembles secondaires par opposition aux ensembles tels qu'ils ont été définis jusqu'ici et que nous appellerons par conséquent ensembles primaires. Les ensembles secondaires seront formés de l'union d'ensembles primaires que nous supposerons disjoints.

Pour exprimer les exercices ayant trait aux permutations avec répétition, nous procéderons comme suit : d'abord, déclarer les p ensembles d'objets identiques $a_1,\ldots,$ a_p en donnant bien sûr pour chacun le nombre d'éléments qu'ils comprennent $(n_1,\ldots, n_p)$; ensuite, déclarer l'ensemble principal comme ensemble secondaire composé des p ensembles primaires.

Revenons à la double distinction (objets pouvant être répétés ou non, objets tous identiques ou tous différents). Nous pouvons donc distinguer 4 types d'ensembles d'objets :

1. les ensembles comprenant des objets pouvant être répétés et tous identiques. Utiliser ce type d'ensemble n'a pas de sens. En effet, si les objets peuvent être répétés, il ne sert à rien de disposer de plusieurs d'entre eux qui seraient identiques : un seul suffit. En outre, un ensemble d'un objet identique est équivalent à un ensemble d'un objet différent. Ce type d'ensemble ne sera donc plus repris par la suite.

2. les ensembles comprenant des objets pouvant être répétés et tous différents.
3. les ensembles comprenant des objets ne pouvant être répétés et tous identiques.
4. les ensembles comprenant des objets ne pouvant être répétés et tous différents.

Par convention, - . lorsqu'un ensemble comprend des objets du type 2, on placera le symbole "*" après le nombre déléments (cf p.29)
. lorsu'un ensemble comprend des objets du type 3, on placera le symbole "„" après le nombre d'éléments
. lorsqu'un ensemble comprend des objets du type 4, on n'ajoutera rien aux déclarations : ce sera l'option par défaut.
- l'ensemble secondaire sera déclaré en donnant son nom suivi du symbole "=", lui-même suivi des noms des ensembles primaires le formant séparés entre eux par le symbole "+" rappelant l'union.
- un ensemble secondaire peut être formé d'ensembles de types différents.

Note : cette convention permet de regrouper en un seul sous-ensemble tous les objets différents, c'est-à-dire les ensembles d'objets identiques a_i dont le cardinal $n_i = 1$

Traduction de l'exercice général.

(nom de l'ensemble d'objets a_1) # n_1";
(nom de l'ensemble d'objets a_2) # n_2";
...
(nom de l'ensemble d'objets a_p) # n_p";
(nom de l'ensemble de cases) # n;
(nom de l'ensemble d'objets) =
(nom de l'ensemble d'objets a_1) +
(nom de l'ensemble d'objets a_2) +
...
(nom de l'ensemble d'objets a_p);
placer n de (nom de l'ensemble d'objets) dans n de (nom de l'ensemble de cases).

Pour utiliser la possibilité signalée dans la note ci-dessus, reformulons la définition de la p.31 : nous disposons de n objets parmi lesquels :

n_1 objets identiques a_1
n_2 objets identiques a_2
...
n_r objets identiques a_r $(n_i > 1 \forall i : 1..r)$
p-r objets différents $a_{r+1} \ldots a_p$
$(a_i \neq a_j \; \forall \; i<j, \; i,j : 1..p)$
$n = \sum_{i=1}^{r} n_i + (p-r)$

Traduction du même exercice en utilisant cette possibilité

(nom de l'ensemble d'objets a_1) # n_1";
(nom de l'ensemble d'objets a_2) # n_2";
...
(nom de l'ensemble d'objets a_r) # n_r";
(nom de l'ensemble d'objets différents) # p-r;
(nom de l'ensemble de cases) # n;
(nom de l'ensemble d'objets) =
(nom de l'ensemble d'objets a_1) +

(nom de l'ensemble d'objets a_2) +
...
(nom de l'ensemble d'objets a_r) +
(nom de l'ensemble d'objets différents);
placer n de (nom de l'ensemble d'objets) dans n de (nom de l'ensemble de cases).

Traduction des exercices 8 et 9 :

Exercice n° 8

```
PAVILLONS # 6;
PLACES-D'UN-PAVILLON # 6;
PLACER 6 DE PAVILLONS DANS 6 DE PLACES-D'UN-PAVILLON.
```

Exercice n° 9

```
PAVILLONS-ROUGES # 2";
PAVILLONS-VERTS # 2";
PAVILLON-BLEU # 1";
PAVILLON-BLANC # 1";
PLACES-D'UN-PAVILLON # 6;
PAVILLONS = PAVILLONS-ROUGES + PAVILLONS-VERTS + PAVILLON-BLEU
+ PAVILLON-BLANC;
PLACER 6 DE PAVILLONS DANS 6 DE PLACES-D'UN-PAVILLON.
```

en utilisant la possibilité décrite ci-dessus :

```
PAVILLONS-ROUGES # 2";
PAVILLONS-VERTS # 2";
AUTRES-PAVILLONS # 2;
PLACES-D'UN-PAVILLON # 6;
PAVILLONS = PAVILLONS-ROUGES + PAVILLONS-VERTS + AUTRES-PAVILLONS;
PLACER 6 DE PAVILLONS DANS 6 DE PLACES-D'UN-PAVILLON.
```

Résumons les concepts vus jusqu'à présent :

. Un exercice consiste en deux parties essentielles : la description des actions à effectuer pour résoudre l'exercice et la description des ensembles manipulés par ces actions qu'on appellera déclarations.

. Nous distinguons deux types de déclarations :
- les déclarations primaires où sont décrits des ensembles primaires disjoints 2 à 2.
- les déclarations secondaires où sont décrits des ensembles secondaires formés de l'union d'ensembles primaires ou secondaires précédemment définis.

. Les ensembles primaires peuvent être de deux catégories : des ensembles d'objets et des ensembles de cases dans lesquelles ces objets sont placés.
Les ensembles primaires d'objets peuvent être de trois types :
- soit un ensemble comprend des objets tous différents pouvant être répétés dans les cases.
- soit un ensemble comprend des objets ne pouvant être répétés et nous les distinguons encore comme suit : soit ils sont tous identiques, soit ils sont tous différents.

Les ensembles primaires de cases peuvent être de deux types :
- soit elles sont distingables.
- soit elles ne le sont pas.

. Une action consiste à choisir n objets parmi m, n cases parmi p et à placer ces n objets choisis dans ces n cases choisies.
Une formule correspond à chaque action, ce qui permet à la machine de calculer la solution de l'exercice ainsi décrit.

2.2.3. Extensions aux concepts de base du langage.

1. Plusieurs actions

Comme nous l'avons vu jusqu'ici, à une action correspond une formule. Avec une seule action, nous ne pouvons donc traduire que des exercices simples (résolus par une formule). Par exemple, cet exercice qui n'est pourtant pas très compliqué ne peut être traduit.

Exercice n° 10
Combien peut-il y avoir de plaques minéralogiques belges du nouveau modèle (3 lettres suivies de 3 chiffres) ?

Il y a deux ensembles d'objets : les lettres au nombre de 26 et les chiffres au nombre de 10. Tous deux comprennent des éléments qui peuvent être répétés. En effet, l'énoncé ne dit pas que les lettres employées doivent être différentes. Il en va de même pour les chiffres.

Il y a deux ensembles de cases : les premiers caractères de la plaque au nombre de 3 et les derniers caractères de la plaque au nombre de 3 également. Elles sont indistingables. D'abord, nous devons choisir trois lettres parmi 26 (26^3) et les placer dans les trois premiers caractères de la plaque. Ensuite, il reste à choisir trois chiffres parmi 10 (10^3) et les placer dans les trois derniers caractères de la plaque. Par conséquent la solution à l'exercice = $26^3 \times 10^3$.

Le raisonnement ainsi décrit comprend deux parties. Nous allons effectuer deux actions pour le décrire, chacune correspondant à une formule. L'opération que nous employons pour trouver la solution à l'exercice est "X". Mais nous pouvons avoir besoin d'autres opérations comme le montrent les exercices ci-dessous.

Exercice n° 11. [3 ex. 2.2.1. (iii) p.28]
Une délégation de 4 lycéens est choisie chaque année pour suivre le congrès annuel de l'Association des Parents d'Elèves. De combien de manières peut-on former la délégation s'il y a 12 lycéens éligibles et si deux d'entre eux sont des frères jumeaux et ne pourront suivre le congrès qu'ensemble ?

2 ensembles d'objets : les frères jumeaux (2), les autres lycéens (10) qui ne peuvent être répétés et qui sont différents.
1 ensemble de cases : les lycéens délégués (4) qui sont indistingables.
Soit les frères jumeaux ne font pas partie de la délégation et nous devons choisir 4 lycéens parmi 10 $\frac{10!}{6!4!}$
Soit ils en font partie et nous devons choisir 2 lycéens parmi 10 $\frac{10!}{8!2!}$

La solution à l'exercice $= \frac{10!}{6!4!} + \frac{10!}{8!2!}$

Exercice n° 12. [3 ex. 2.2.1. (ii) p.28]
Une délégation de 4 lycéens est choisie chaque année pour suivre le congrès annuel de l'Association des Parents d'Elèves. De combien de manières peut-on former la délégation s'il y a 12 lycéens éligibles et si deux d'entre eux refusent de suivre le congrès ensemble ?

Mêmes ensembles que pour l'exercice n° 11 sauf que le premier ensemble comprend cette fois deux ennemis.
D'abord, supposons qu'il n'y ait pas de contraintes : nous devons alors choisir 4 lycéens parmi 12 $\frac{12!}{8!4!}$

Nous devons retrancher les cas où les ennemis participeraient ensemble à la délégation : les deux autres lycéens délégués sont alors choisis parmi 10 $\frac{10!}{8!2!}$

La solution à l'exercice $= \frac{12!}{8!4!} - \frac{10!}{8!2!}$

Un exercice est donc décrit par une suite d'actions, chacune correspondant à une formule. La solution finale est obtenue en appliquant les opérateurs X, + et - aux formules des différentes actions. Nous devons donc signaler ces opérateurs dans le langage pour que la machine puisse calculer la solution finale et l'expliquer.

Par convention,
- lorsque l'opérateur X doit être appliqué aux formules de deux actions consécutives, on n'ajoutera rien à la forme actuelle des actions : ce sera l'option par défaut.
- lorsque l'opérateur + doit être appliqué aux formules de deux actions consécutives, on ajoutera le nouveau mot réservé "ou" (au sens du ou exclusif en logique) avant le mot réservé "placer".
- lorsque l'opérateur - doit être appliqué aux formules de deux actions consécutives, on ajoutera le nouveau mot réservé "sans" avant le mot réservé "placer".
- pour calculer la solution finale, nous utilisons les règles de priorité d'opérateurs habituelles en mathématiques. Le "ou" et le "sans" seront donc de priorité plus basse que l'option par défaut.

Il nous reste à traduire les exercices présentés ci-dessus.

<u>Exercice n° 10</u>.

```
LETTRES # 26*;
CHIFFRES # 10*;
PREMIERS-CARACTERES-DE-PLAQUE # 3;
DERNIERs-CARACTERES-DE-PLAQUE # 3;
PLACER 3 DE LETTRES DANS 3 DE PREMIERS-CARACTERES-DE-PLAQUE;
PLACER 3 DE CHIFFRES DANS 3 DE DERNIERS-CARACTERES-DE-PLAQUE.
```

Exercice n° 11.

```
FRERES-JUMEAUX # 2;
AUTRES-LYCEENS # 10;
LYCEENS-DELEGUES # 4!;
PLACER 4 DE AUTRES-LYCEENS DANS 4 DE LYCEENS-DELEGUES;
OU PLACER 2 DE FRERES-JUMEAUX DANS 2 DE LYCEENS-DELEGUES;
PLACER 2 DE AUTRES-LYCEENS DANS 2 DE LYCEENS-DELEGUES.
```

Exercice n° 12.

```
ENNEMIS # 2;
AUTRES-LYCEENS # 10;
LYCEENS-DELEGUES # 4!;
LYCEENS-ELIGIBLES = ENNEMIS + AUTRES-LYCEENS;
PLACER 4 DE LYCEENS-ELIGIBLES DANS 4 DE LYCEENS-DELEGUES;
SANS PLACER 2 DE ENNEMIS DANS 2 DE LYCEENS-DELEGUES;
PLACER 2 DE AUTRES-LYCEENS DANS 2 DE LYCEENS-DELEGUES.
```

Note : Les actions seront traitées séparément : à chaque action, correspond une solution d'action. Ceci a pour conséquence que les liens intuitifs que l'on pourrait concevoir entre les actions ne sont pas traités comme tels par le système. Par exemple, les dernières lignes des exercices 11 et 12 ne correspondent pas à l'énoncé : nous voulons en effet signifier que nous prenons les deux autres lycéens-délégués. Or, le système, oubliant ce qui a été fait lors de l'action précédente pourrait choisir lors de la dernière action des lycéens-délégués déjà choisis. Le résultat numérique sera cependant fortuitement identique. Nous verrons plus loin (p.50) la traduction correcte de ces deux exercices. Notons

enfin que le système admettrait que l'on remplace le nombre 2 par le nombre 3 lors des dernières actions des exercices 11 et 12. L'emploi de plusieurs actions doit donc être très prudent et doit tenir compte du fait qu'elles seront traitées séparément.

2. Extensions des déclarations.

Jusqu'à présent, les déclarations primaires comportent :

- le nom de l'ensemble pour que le système fasse le lien avec l'énoncé en langage naturel;
- le nombre d'éléments appartenant à l'ensemble pour que le système calcule la solution;
- le type de l'ensemble également pour que le système calcule la solution.

Quelquefois, le nom de l'ensemble peut s'avérer insuffisant pour que l'élève comprenne les explications générées par le programme : en effet, lorsqu'un enseignant explique une solution, il est souvent amené à citer des éléments de l'ensemble :

- soit il désire énumérer les différents arrangements, permutations ou combinaisons suivant les cas.
- soit il utilise des exemples parlants pour sa classe (exemple : plutôt que de parler d'un ensemble de 3 élèves, il parle de ses élèves Pierre, Jacques et Jean).
- soit il utilise des identificateurs désignant les éléments de l'exercice (exemple : [3 p.28 ex. 2.2.1. (iii)] Soient C et D les lycéens qui sont frères jumeaux ...)

C'est pourquoi, sous peine de perdre une dimension importante, les explications générées par le programme devraient permettre de citer des éléments. Pour ce faire, nous devons bien sûr les inclure dans la description de l'exercice. C'est pourquoi deux façons de déclarer les éléments appartenant à un ensemble seront permises : en donnant leur nombre et en les citant. Dans cette éventualité, les éléments ne

pourront évidemment pas être tous identiques puisqu'ils sont chacun identifiés par un nom différent.

Deux moyens de nommer les éléments sont permis : soit par 1 lettre ou 1 chiffre, soit par un nom quelconque non restreint à un caractère. Dans le premier cas, une facilité supplémentaire est offerte : l'on peut citer uniquement le premier et le dernier élément de l'ensemble. Pour que la machine sache quels sont les autres éléments appartenant à l'ensemble, nous devons définir les éléments susceptibles d'appartenir à un tel intervalle ainsi qu'un ordre sur ces éléments. Nous avons donc choisi d'utiliser les 26 lettres et les 10 chiffres, au total 36 éléments. L'ordre sur ces éléments sera défini comme suit : le rang de tout chiffre est inférieur à celui de toute lettre, l'ordre sur les chiffres est l'ordre arithmétique (de 0 à 9) et l'ordre sur les lettres est l'ordre alphabétique.

Par convention,
- lorsqu'un ensemble sera déclaré en citant tous ses éléments, la déclaration aura la forme suivante :
  (nom de l'ensemble) = ((nom du 1er élément), (nom du 2eme élément), ..., (nom du dernier élément))
- lorsqu'un ensemble sera déclaré en citant le premier et le dernier de ses éléments, la déclaration aura la forme suivante :
  (nom de l'ensemble) = ((nom du 1er élément) .. (nom du dernier élément))

Voici maintenant un exemple décrivant 5 façons de signaler qu'un ensemble possède 3 éléments :

élèves 3
élèves = (Pierre, Jacques, Jean)
élèves = (1..3)
élèves = (A..C)
élèves = (8..A) est équivalent à (8, 9, A)

Note : Dorénavant, pour différencier un nom d'ensemble et un nom d'élément, par convention, les noms d'ensembles seront préfixés par "%".

3. Création de sous-ensembles.

Une action consiste à choisir n objets parmi tous les éléments de l'ensemble d'objets dont le cardinal est m et de même pour les cases et à placer ces objets choisis dans les cases choisies. Il serait souhaitable de choisir n éléments non parmi tous les éléments de l'ensemble mais parmi une partie de celui-ci. Jusqu'ici, chaque fois que c'était possible, ce problème a été résolu en divisant cet ensemble d'emblée lors des déclarations (exemple : dans l'exercice 10, nous avons déclaré deux ensembles de cases : premiers et derniers caractères de la plaque alors qu'il serait plus heureux de ne plus en faire qu'un seul nommé caractères de la plaque).

Nous allons maintenant résoudre ce problème plus élégamment en créant des sous-ensembles lors des actions. Le sous-ensemble dans lequel nous pourrons choisir les éléments pourra être défini de deux manières différentes :

. tout d'abord, comme résultat de la soustraction d'un sous-ensemble créé de l'ensemble principal. (a)

. ensuite, plus simplement, comme un sous-ensemble créé (b)

(exemple : dans l'exercice 10, supposons que nous avons déclaré un ensemble caractères de la plaque et que nous désirons, pour la première action, choisir 3 cases parmi les 3 premières pour placer les lettres et non pas parmi la totalité des cases). Nous pouvons dire que

(a) nous choisissons parmi l'ensemble des cases duquel on a retiré les 3 dernières.

(b) nous choisissons parmi les 3 premières cases de l'ensemble.

Par convention, . définir un sous-ensemble de la manière (a) sera fait comme suit
n de (nom de l'ensemble) M (désignation du sous-ensemble). ("M" rappelant le signe moins).
. définir un sous-ensemble de la manière (b) sera fait comme suit
m de (nom de l'ensemble) R (désignation du sous-ensemble). ("R" signifiant "restreint à" : l'on restreint l'ensemble au sous-ensemble qui y est inclu).

Note : L'utilisation simultanée de "M" et de "R" n'est pas interdite pour un même ensemble lors d'une même action. Dans ce cas, l'on placera "R" avant "M" et les deux sous-ensembles seront considérés comme disjoints.

Lorsque le nombre d'éléments parmi lesquels nous devons choisir sera égal au nombre d'éléments à choisir, nous pourrons remplacer ce dernier par le mot réservé tout.

Nous distinguons 3 types de sous-ensembles :
A. suite ordonnée : le sous-ensemble est désigné en donnant le rang des éléments qui lui appartiennent (exemple : i^{eme} élément, j^{eme} élément, etc...). (1)
Il est aussi permis de le désigner en signifiant que nous prenons les éléments compris entre le i^{eme} et le j^{eme}. (2)

Par convention, . la possibilité (1) sera traduite comme suit : [i, j, ...]
. la possibilité (2) sera traduite comme suit : [i .. j]

Traduisons à nouveau l'exercice n° 10.

```
%LETTRES # 26*;

%CHIFFRES # 10*;

%CARACTERES-DE-LA-PLAQUE # 6;

PLACER 3 DE %LETTRES DANS  3    DE %CARACTERES-DE-LA-PLAQUE R [1..3]  :
                          TOUT                               R [1,2,3]
                                                             M [4..6]
                                                             M [4,5,6]

PLACER 3 DE %CHIFFRES DANS  3    DE %CARACTERES-DE-LA-PLAQUE R [4..6]
                           TOUT                               R [4,5,6]
                                                              M [1..3]
                                                              M [1,2,3]
```

N.B. : Nous pouvons employer <u>tout</u> car nous choisissons 3 cases parmi 3.

<u>B. suite</u> : le sous-ensemble est désigné en citant les éléments qui lui appartiennent (exemple : Jacques, Jean) (1)
Il est aussi permis de le désigner en signifiant que nous prenons les éléments qui ont été déclarés entre un tel élément et un tel élément. (2)

Par convention, . la possibilité (1) sera traduite comme suit :
((nom du 1er élément du sous-ensemble),
(nom du 2eme élément du sous-ensemble), ...)
. la possibilité (2) sera traduite comme suit :
((nom du 1er élément du sous-ensemble)..
(nom du dernier élément du sous-ensemble))

Exemple : supposons un ensemble de 16 enfants et un ensemble de 4 ballons déclarés comme suit :

```
% enfants = (0..F)
% ballons = (jaune, vert, bleu, rouge),
```

nous pouvons créer les sous-ensembles suivants :
pour les enfants : (A,B,C); (3..5); (F)
pour les ballons : (jaune .. bleu) = (jaune, vert, bleu)

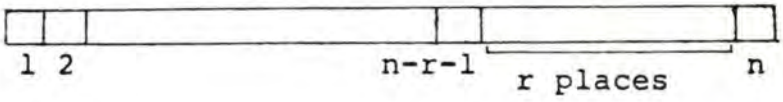
Note : Une suite ne peut être créée si l'ensemble qui est

sensé la contenir a été déclaré sans citer les éléments qui appartiennent (par conséquent en donnant uniquement le nombre d'éléments)
Par contre, une suite ordonnée peut être créée que les éléments lui appartenant aient été cités ou pas.

C. suite variable : le sous-ensemble est désigné en fonction du choix de n éléments parmi m qui a été effectué lors d'une action précédente. La plupart du temps, plusieurs façons de choisir ces n éléments sont possibles. Il seront par conséquent représentés par une variable destinée à couvrir tous ces choix différents. Voyons un exercice où cette possibilité est indispensable.

Exercice N° 13. [6 ex. 7]
Deux amis se trouvent dans la queue à l'entrée d'un restaurant self-service. Sachant que la queue comporte n personnes alignées, combien y a-t-il de cas où ils sont séparés par r personnes exactement ?

résolution :



1) Nous allons d'abord voir où peut se placer l'ami qui sera le premier dans la queue : il ne peut se placer plus loin que la (n-r-1) ème place; sinon, l'autre ne pourra être séparé de lui de r places. En résumé, nous plaçons 1 des 2 amis dans 1 des n-r-1 premières places.
2) Nous plaçons l'autre ami (r + 1) places plus loin.
3) Enfin, les (n-2) autres personnes sont placées dans les (n-2) places restantes.

Il nous reste à calculer de combien de façons chacune de ces actions peut être faite.

Dans la deuxième étape de résolution, nous parlons de l'autre ami en faisant référence à celui choisi lors de la première étape sans savoir duquel il s'agit. Par conséquent,

nous ne pouvons utiliser ni une suite ordonnée (impossible de préciser s'il s'agit du premier ou du deuxième) ni une suite (même en les nommant par exemple A et B, impossible de savoir s'il s'agit de A ou de B).

Pour résoudre ce problème, nous nommerons l'ami choisi lors de la première étape à l'aide d'une variable (Nous plaçons 1 des 2 amis, soit x l'ami choisi, dans...). Dans la seconde étape de résolution, nous choisirons parmi l'ensemble des amis duquel nous aurons au préalable retranché x (Cette formulation est donc l'équivalent de "l'autre ami" que nous avons employé lors de l'explication de résolution en langage naturel). Le sous-ensemble ainsi créé ne possède donc plus qu'un élément.

Deux phases seront donc nécessaires pour créer une suite variable :

1. Tout sous-ensemble de n éléments choisis parmi m susceptible de servir de référence pour créer une suite variable sera nommé.
2. Lors des actions suivantes, nous pourrons utiliser ce nom pour créer une suite variable.

Revenons à la résolution de l'exercice n° 13 pour l'exprimer de façon encore plus proche de ce que nous avons défini.

1) nous plaçons 1 des 2 amis, soit x l'ami choisi dans une des n-r-1 premières places, soit i la place choisie.
2) nous plaçons l'ami qui n'est pas x dans la place i + (r + 1), soit j cette place.
3) nous plaçons les (n-2) autres personnes dans les places qui ne sont ni i, ni j.

Une suite variable pourra prendre deux formes :

a. elle peut être composée d'une suite de sous-ensembles choisis. (cf ami qui n'est pas x; places qui ne sont ni i, ni j)
b. elle peut être définie par une position par rapport à un

sous-ensemble choisi.
(cf la place i + (r + 1))

Par convention, . la phase 1. sera traduite comme suit : après les mots réservés placer ou dans (selon qu'il s'agit d'objets ou de cases), on ajoutera (nom du sous-ensemble choisi) =...
. définir une suite variable de la forme a. se fera comme suit : ((nom d'un sous-ensemble choisi) + (nom d'un sous-ensemble choisi)+...)
. définir une suite variable de la forme b. se fera comme suit : (nom d'un sous-ensemble choisi) (signe) (nombre) , le signe étant soit +, soit -.
. le nom d'un sous-ensemble choisi sera aussi préfixé du symbole "%".

Traduction de l'exercice n° 13

Notons à propos de cet exercice que le système permet d'utiliser des variables. Le résultat sera donc une expression.

```
%AMIS # 2;
%AUTRES-PERSONNES # (N-2);
%PLACES # 2;
PLACER %X = 1 DE %AMIS DANS %I = 1 DE %PLACES R [1..(N-R-1)];
PLACER 1 DE %AMIS M (%X) DANS %J = 1 DE %PLACES R (%I + (R+1));
PLACER TOUT DE %AUTRES-PERSONNES DANS TOUT DE %PLACES M (%I + %J)
```

Traduction de l'exercice n° 11.

```
%FRERES-JUMEAUX # 2;
%AUTRES-LYCEENS # 10;
%LYCEENS-DELEGUES # 4!;
PLACER 4 DE %AUTRES-LYCEENS DANS TOUT DE %LYCEENS-DELEGUES;
OU PLACER TOUT DE %FRERES-JUMEAUX DANS %I = 2 DE %LYCEENS-DELEGUE
PLACER 2 DE %AUTRES-LYCEENS DANS TOUT DE %LYCEENS-DELEGUES M (%I)
```

Traduction de l'exercice n° 12.

```
%ENNEMIS # 2;
%AUTRES-LYCEENS # 10;
%LYCEENS-DELEGUES # 4!;
%LYCEENS-ELIGIBLES = %ENNEMIS + %AUTRES-LYCEENS;
PLACER 4 DE %LYCEENS-ELIGIBLES DANS TOUT DE %LYCEENS-DELEGUES;
SANS PLACER TOUT DE %ENNEMIS DANS %I = 2 DE %LYCEENS-DELEGUES;
PLACER 2 DE %AUTRES-LYCEENS DANS TOUT DE LYCEENS-DELEGUES M (%I).
```

4. Partitions.

Jusqu'à présent, l'application de l'ensemble des objets choisis vers l'ensemble des cases choisies est bijective (tout objet choisi est placé dans une et une seule case et toute case choisie contient un et un seul objet). Certains problèmes, nous pensons tout spécialement aux problèmes de partition 3 p.22,29 et 30 , nous amènent à envisager des applications non injectives (plusieurs objets choisis pourraient être placés dans une même case) mais toujours surjectives (toute case choisie contenant au moins un objet).

Exercice n° 14. 3 ex. [2.26 p 30]
Une classe comporte 12 élèves. De combien de manières ces 12 élèves peuvent-ils subir 3 examens différents, sachant que 4 élèves subissent le même examen ?
Tentons de formaliser cet énoncé : nous avons un ensemble d'objets, les élèves au nombre de 12 et un ensemble de cases, les examens au nombre de 3.
Les objets sont différents et ne peuvent être répétés. Les cases sont indistingables. Nous devons choisir 12 élèves parmi 12 et les placer dans 3 cases choisies parmi 3. Tout ceci peut être exprimé dans le langage mais pour être complet, nous devons préciser combien d'objets contient chacune de ces cases.

Le problème consiste donc en l'expression du nombre d'objets que contient chaque case. Supposons que nous disposons de n objets choisis et que nous désirons les placer dans r cases choisies ($n>r$), une case contenant n_1 objets, une deuxième n_2 objets, ..., la $r^{ème}$ $n_r \left(\sum_{i=1}^{r} n_i = n \right)$. Pour ne pas citer ainsi tous les couples case-nombre d'objets contenus, nous avons prévu une expression regroupant les cases contenant le même nombre d'objets. Par conséquent nous avons r_1 cases contenant chacune n_1 objets, r_2 cases contenant chacune n_2 objets, ..., r_s cases contenant chacune n_s objets $(n_i \neq n_j \; \forall \; i<j;\; i,j = 1...s) \left(\sum_{i=1}^{s} r_i = r \right) \left(\sum_{i=1}^{s} (r_i * n_i) = n \right)$.
Suivant que ces cases sont indistingables ou non, 3 donne à ces deux cas le nom de partition ordonnée et non ordonnée.

Par convention, avant le séparateur de fin de ligne (";" ou "."), on ajoutera le caractère "," suivi d'une série de couples (nombre de cases contenant le même nombre d'objets), (nombre d'objets) . Pour séparer les deux nombres du couple, nous placerons le symbole ":" entre eux et pour séparer les couples, nous laisserons un espace entre eux.

Exemples.

1) Exercice n° 14.

```
%ELEVEs # 12;
%EXAMENS # 3;
PLACER TOUT DE %ELEVES DANS TOUT DE %EXAMENS , 3:4.
```

2) Exercice n° 15. [3 ex. 2.67 p 33]
De combien de manières peut-on faire une partition d'une assemblée de 14 personnes en 6 comités dont 2 comprennent 3 membres et les autres 2 ?

```
%PERSONNES # 14;
%COMITES # 6;
PLACER TOUT DE %PERSONNES DANS TOUT DE %COMITES , 2:3 4:2.
```

Explication : Contrairement à l'autre exercice, les cases sont indistingables. En effet, deux comités contenant le même nombre de personnes ne sont pas différents alors que tous les examens sont différents.

5. Nouveau type d'action.

En étendant encore plus les concepts vus ci-dessus, nous allons envisager des applications de l'ensemble des objets choisis vers l'ensemble des cases choisies qui ne soient ni surjectives, ni injectives. Une case choisie pourra contenir de 0 à n objets (n étant le nombre d'objets choisis). Nous n'allons cependant pas préciser combien d'objets contient chacune des cases, l'objectif étant cette fois de dénombrer toutes les façons de répartir n objets parmi r cases. Voyons à l'aide d'un exemple quel type d'exercice ce nouveau concept permet de résoudre.

Exercice n° 16. [6 ex.15 1°]
On dispose de 5 outils et de 7 casiers susceptibles de les recevoir. On suppose que chaque casier peut contenir les 5

outils. Déterminer le nombre de façons de placer les 5 outils dans les 7 casiers d'une façon quelconque.
Résolution : cet exercice est assez simple à résoudre. En effet considérant le premier outil, il y a 7 façons de le placer; pour le second, il en va de même puisque l'on peut placer plusieurs outils dans un même casier; et ainsi de suite pour les autres. Nous avons donc 7^5 façons de placer les outils, ce qui correspond à la formule des arrangements avec répétition. Nous pouvons donc traduire cet énoncé comme suit :

```
%CASIERS # 7*;
%OUTILS # 5;
PLACER 5 DE %CASIERS DANS TOUT DE %OUTILS.
```

Cette expression est assez peu heureuse et est trop éloignée de la réalité pour que nous la retenions.

Pour résoudre ce problème, nous allons définir un nouveau type d'action qui (le premier étant "placer") signifiera que l'on désire dénombrer le nombre de façons de répartir ou de distribuer n objets dans r cases, une case pouvant contenir de 0 à n objets.

Par convention, pour décrire ce type d'action, on remplacera le mot réservé placer par le mot réservé répartir.

Traduisons l'exercice 16. :

```
%CASIERS # 7;
%OUTILS # 5;
REPARTIR TOUT DE %OUTILS DANS TOUT DE %CASIERS.
```

6. Nouveaux types d'ensembles de cases.

Nous avons jusqu'ici distingué deux types d'ensembles de cases, selon que leur ordre a ou n'a pas d'importance. Voyons ce que cela signifie si nous disposons de n objets choisis placés dans r cases choisies :

1) Si les cases sont distingables, permuter les objets de deux cases quelconques donne un cas supplémentaire dans la solution.
2) Si les cases sont indistingables, permuter les objets dans toutes les cases ne donne pas de cas supplémentaire dans la solution.

Entre ces deux extrémités, nous allons distinguer d'autres types

3) Enlever les objets de la case i (i=1..r-1) pour les placer dans la case i + 1 (les objets de la case r étant transférés vers la case 1) ne donne pas de cas supplémentaire pour la solution. Pour cela, les r cases doivent évidemment contenir le même nombre d'objets. L'exemple type de ceci, ce sont les permutations circulaires dont voici un exemple.

Exemple n° 17. [3 2.5 (ii) p.25]
De combien de façons différentes peut-on répartir un groupe de 7 personnes autour d'une table ronde ?
Ayant placé les 7 personnes, si on les fait bouger chacune d'une chaise, on obtient la même disposition.

4) Enlever les objets de la case i (i:1..r) pour les placer dans la case r-i+1 ne donne pas de cas supplémentaire. Donc l'objet de la première case peut être déplacé dans la dernière case, celui de la deuxième dans l'avant-dernière, ..., celui de la dernière dans la première. Pour cela, ces deux cases doivent contenir le même nombre d'objets.

5) Effectuer les deux permutations décrites en 3) et en 4)

ne donne pas de cas supplémentaire. Voyons un exemple :

Exercice n° 18. [7 ex. 2° p. 8]

Soit dans un plan, n points A1, A2, A3, ..., An tels que trois points quelconques d'entre eux ne soient pas alignés. Combien peut-on former de polygones ayant pour sommets p de ces points ?

Résolution : nous avons vu un ensemble de n objets (les points) parmi lesquels nous en choisissons p pour les placer dans p cases choisies parmi p (les sommets).

Soit le polygone formé des points A1, A2, ..., Ap, il n'est pas différent du polygone A2, A3, ..., Ap, A1 ni du polygone Ap, Ap-1, ..., A2, A1.

Les conventions pour les types d'ensemble 1) et 2) ont déjà été présentés p. .

Lorsque les ensembles de cases seront du type 3), on ajoutera le symbole "," après le nombre de cases.

Lorsque les ensembles de cases seront du type 4), on ajoutera le symbole ":" après le nombre de cases.

Lorsque les ensembles de cases seront du type 5), on ajoutera le symbole "&" après le nombre de cases.

Traduisons maintenant les exemples donnés ci-dessus :

Exercice n° 17

```
%PERSONNES # 7;
%PLACES # 7';
PLACER TOUT DE %PERSONNES DANS TOUT DE %PLACES.
```

Exercice n° 18

```
%POINTS # N;
%SOMMETS # P&;
PLACER P DE %POINTS DANS TOUT DE %SOMMETS.
```

7. Précision d'ensembles d'objets.

Nous allons maintenant présenter un outil permettant de faciliter la rédaction d'exercices dans lesquels les cases choisies appartiennent à un seul ensemble et contiennent des objets choisis appartenant à des ensembles différents. Voici un exemple de ce type d'exercice :

Exercice n° 19 [3 ex. 2.55 (ii) p.33]

Une classe comporte 9 garçons et 3 filles. De combien de manières le professeur peut-il faire un choix de 4 élèves si ces choix doivent comporter au moins une fille ?

un ensemble de cases : les élèves choisis au nombre de 4

deux ensembles d'objets : les garçons au nombre de 9 et
les filles au nombre de 3

Les objets de deux ensembles différents vont donc être placés dans un seul ensemble de cases.

Cet énoncé peut être traduit comme suit dans le langage si l'on utilise uniquement les concepts présentés jusqu'ici.:

Exercice n° 19

```
%GARCONS # 9;
%FILLES # 3;
%ELEVES-CHOISIS # 4';
PLACER 1 DE %FILLES DANS %I = 1 DE %ELEVES-CHOISIS;
PLACER 3 DE %GARCONS DANS TOUT DE %ELEVES-CHOISIS M (%I);
OU PLACER 2 DE %FILLES DANS %J = 2 DE %ELEVES-CHOISIS;
PLACER 2 DE %GARCONS DANS TOUT DE %ELEVES-CHOISIS M (%J);
OU PLACER TOUT DE %FILLES DANS %K = 3 DE %ELEVES-CHOISIS;
PLACER 1 DE %GARCONS DANS TOUT DE %ELEVES-CHOISIS M (%K).
```

Cette rédaction est assez fastidieuse pour un énoncé pourtant simple. En précisant que l'on choisit 4 élèves parmi 9 garçons et 3 filles avec la contrainte que le choix doit comporter au moins une fille, il serait possible à l'ordina-

teur de générer ces différentes lignes d'exercice où l'on distingue les cas où il y a soit 1 fille et 3 garçons dans les élèves choisis, soit 2 filles et 2 garçons, soit 3 filles et 1 garçon.

Nous allons donc exposer un moyen d'écrire ceci beaucoup plus simplement. Comme auparavant, nous allons déclarer p ensembles primaires d'objets ($p \geq 2$) et un ensemble primaire de cases. Par contre, nous allons maintenant déclarer un ensemble secondaire qui contiendra les p ensembles primaires d'objets que nous supposerons de cardinal m_1, ..., m_p. L'action consistera à choisir n objets de l'ensemble secondaire et à les placer dans l'entièreté de l'ensemble de cases qui devra par conséquent contenir n éléments. Nous préciserons aussi comment choisir ces n objets c'est-à-dire combien en prendre dans chaque ensemble primaire. L'écriture permettra de n'indiquer que ce qui est nécessaire et suffisant pour que la machine "comprenne" ce que nous signifions. Donnons un exemple : supposons deux ensembles primaires %A et %B composant l'ensemble secondaire %C. Si nous choisissons n objets de %C en précisant que parmi les n, nous en prenons r en provenance de %A ($r \leq n$), la machine "saura" aussi que nous en prenons n-r en provenance de %B. De plus, le nombre d'éléments choisis dans l'ensemble composant l'ensemble secondaire pourra être un minimum, un maximum ou le nombre exact. Reprenons notre exemple ci-dessus et supposons que nous prenons au minimum r objets en provenance de %A. Cela signifiera que nous tenons compte des cas où on en prend r de %A et n-r de %B, r+1 de %A et n-(r+1) de %B,... n de %A et 0 de %B. (à condition que l'ensemble %A contienne au moins n éléments)

Par convention, . après l'ensemble d'objets nous ajouterons le mot réservé <u>avec</u> suivi d'une série de ((nombre d'éléments à choisir) <u>de</u> (nom de l'ensemble dans lequel ces éléments sont à choisir)) séparés entre eux par le mot

réservé et;

. si le nombre d'éléments à choisir est maximum, on lui acollera le signe "+" (au sens de "au plus"); par contre si le nombre d'éléments à choisir est minimum, on lui acollera le signe "-" (au sens de "au moins").

Traduction de l'exercice n° 19.

```
%GARCONS # 9;

%FILLES # 3;

%ELEVES-CHOISIS # 4;

%ELEVES = %GARCONS + %FILLES;

PLACER 4 DE %ELEVES AVEC 1 - DE %FILLES DANS TOUT DE %ELEVES-CHOISIS
```

Note : Certains outils du langage ne pourront être utilisés simultanément dans une même action tout d'abord parce que, parmi les innombrables exercices que nous avons pu traiter, il n'était pas utile de le faire et ensuite parce qu'une telle implémentation serait particulièrement complexe. Voici la liste des outils incompatibles.

- Les partitions, le type d'action "répartir" et la précision d'ensembles d'objets sont mutuellement incompatibles.
- Les partitions ne s'utilisent qu'avec des ensembles d'objets différents et ne pouvant être répétés et des ensembles de cases, soit indistingables, soit totalement distingables.
- Le type d'action "répartir" ne s'utilise qu'avec des ensembles d'objets différents et ne pouvant être répétés et des ensembles de cases totalement distingables.
- La précision d'ensembles d'objets ne s'utilise qu'avec des ensembles d'objets différents et ne pouvant être répétés et des ensembles de cases indistingables. De plus, nous ne pouvons utiliser un minimum et un maximum dans une même action.

- Les ensembles secondaires ne comprennent que des objets qui ne peuvent être répétés et qui sont déclarés sans être cités.
- Les objets pouvant être répétés ne peuvent être placés que dans des cases indistingables ou totalement distingables.
- Les objets identiques ne peuvent être placés que dans des cases totalement distingables.

Note sur le contenu des annexes concernant le langage.

En annexe 1, nous avons inclu la grammaire du langage qui définit de façon rigoureuse et précise la syntaxe du langage.

L'annexe 2 consiste en la liste des erreurs. Celles dont le numéro est ≤ 100 sont des erreurs de syntaxe, celles dont le numéro est > 100 et ≤ 200 sont des erreurs de sémantique, celles dont le numéro est > 200 et ≤ 300 signalent que l'on a utilisé des outils du langage incompatibles, celles dont le numéro est > 300 et ≤ 400 signalent que l'on a dépassé une contrainte du système, celles dont le numéro > 400 signalent que l'on a utilisé une possibilité du système non encore implémentée.

L'annexe 3 reprend des exercices traduits dans le langage.

2.3. Formules applicables aux exercices d'analyse combinatoire traduits dans le langage.

Donner les formules applicables à une action suffit puisque nous connaissons (p 38 à 42) les opérateurs à utiliser pour trouver le résultat de l'exercice complet. Dans la première partie, nous allons présenter les formules des actions n'utilisant pas les outils particuliers que sont la partition, le type d'action "répartir" et la précision qui feront l'objet de la seconde partie.

1ere partie

Puisque nous n'utilisons ne la partition ni le type d'action "répartir", le nombre de cases choisies égale le nombre d'objets choisis et nous pouvons dire que toute action consiste à choisir n objets parmi m et à les placer dans n cases choisies parmi p. Nous allons particulariser ceci en fonction du type des ensembles d'objets et de cases.

1. objets différents et ne pouvant être répétés.
   a. cases totalement distingables.
      . Choisissons d'abord n objets parmi m sans tenir compte de l'ordre dans lequel ce choix est effectué. Cela peut se faire de $\frac{m!}{(m-n)!\ n!}$ façons possibles.
      . Choisissons ensuite n cases parmi p sans tenir compte de l'ordre dans lequel ce choix est effectué. Cela peut se faire de $\frac{p!}{(p-n)!\ n!}$ façons possibles
      . Enfin, disposant de n objets et de n cases, il reste à placer les uns dans les autres en tenant compte du fait que les cases sont distingables. Cela peut se faire de n! façons.
      . Cela donne au total $\frac{m!}{(m-n)!\ n!} \times \frac{p!}{(p-n)!\ n!} \times n!$

Note : lorsque p = n, nous obtenons la formule des arrangements sans répétition, ce qui confirme notre traduction de la p. 24.
Si de plus m = n, nous obtenons la formule des permutations sans répétition, ce qui confirme notre traduction de la p. 24.

   b. cases indistingables (!)
      . Lorsque les cases sont indistingables, choisir n cases parmi p ne peut se faire que d'une seule façon. En effet, les cases n'étant pas différentes, peu importe celles qui sont choisies.
      . Il reste donc à choisir n objets parmi m sans tenir compte de l'ordre dans lequel ce choix est effectué. Cela peut se faire de $\frac{m!}{(m-n)!\ n!}$ façons possibles.

Note : nous obtenons bien la formule des combinaisons sans répétition, ce qui confirme notre traduction de la p. 28.

c. Ensemble de cases du type 3 p.54 (')

. Les deux premières parties du raisonnement sont semblables à celles de a.

. Disposant de n objets et de n cases, il reste à placer les uns dans les autres en tenant compte du fait que déplacer chaque objet d'une case dans le même sens ne donne pas de cas supplémentaire. Puisqu'il y a n déplacements de ce type, nous devons donc diviser le résultat du cas a. par n, ce qui nous donne

$$\frac{n!}{n} = (n-1)!$$

. Cela donne au total $\frac{m!}{(m-n)!\ n!} \times \frac{p!}{(p-n)!\ n!} \times (n-1)!$

d. Ensemble de cases du type 4 p.54 (:)

. Les deux premières parties du raisonnement sont semblables à celles de a.

. Disposant de n objets et de n cases, il reste à placer les uns dans les autres en tenant compte du fait que déplacer les objets de la case i vers la case n-i+1 ne donne pas de cas supplémentaire. Puisqu'il y a deux déplacements de ce type, (si $n \geqslant 2$), nous devons donc diviser le résultat du cas a. par 2, ce qui nous donne $\frac{n!}{2}$ si $n \geqslant 2$ et 1 si n=1.

. Cela donne au total $\frac{m!}{(m-n)!\ n!} \times \frac{p!}{(p-n)!\ n!} \times \frac{n!}{2}$ si $n \geqslant 2$

m X p si n=1

e. Ensemble de cases du type 5 p.54 (&)

. Les deux premières parties du raisonnement sont semblables à celles de a.

. Disposant de n objets et de n cases, il reste à placer les uns dans les autres en tenant compte du fait que les deux déplacements décrits ci-dessus ne donnent pas de cas supplémentaire. Il y a donc deux m déplacements qui donnent le même cas (si $n \geqslant 3$). Nous devons

donc diviser le résultat du cas a. par 2n, ce qui nous donne $\frac{n!}{2n}$ ou $\frac{(n-1)!}{2}$ si $n \geqslant 3$ et 1 si n=1 ou 2

. Cela donne au total

$$\frac{m!}{(m-n)!\ n!} \times \frac{p!}{(p-n)!\ n!} \times \frac{(n-1)!}{2} \quad \text{si } n \geqslant 3$$

$$\frac{m \times (m-1)}{2} \times \frac{p \times (p-1)}{2} \quad \text{si } n = 2$$

$$m \times p \quad \text{si } n = 1$$

2. objets pouvant être répétés (~~*~~)

Note : comme signalé p.59 , les cases ne peuvent être du type 3, 4 ou 5.

a. cases distingables.

. Choisissons d'abord n cases parmi p sans tenir compte de l'ordre dans lequel ce choix est effectué. Cela peut se faire de $\frac{p!}{(p-n)!\ n!}$ façons possibles.

. Dans chacune de ces n cases, nous pouvons placer 1 des m objets.

Cela peut donc se faire de m^n façons possibles.

. Cela donne au total $\frac{p!}{(p-n)!\ n!} \times m^n$

Note : lorsque p = n, nous obtenons la formule des arrangements avec répétition, ce qui confirme notre traduction de la p. 30

b. cases indistingables.

. La remarque de 1.b. étant encore d'application, il reste à choisir n objets pouvant être répétés parmi m sans tenir compte de l'ordre dans lequel ce choix est effectué.

Cela peut se faire de $\frac{(m+n-1)!}{(m-1)!\ n!}$ façons possibles.

(formule des combinaisons avec répétition)

3. ensemble contenant au moins deux objets identiques.

Note : comme signalé p.59 , les cases doivent être totalement distingables.

Supposons que parmi les m objets susceptibles d'être choisis nous avons m_1 objets identiques a_1, m_2 objets identiques a_2, ..., m_r objets identiques a_r, m_{r+1} objets différents $(\sum_{i=1}^{r+1} m_i = m) \cdot (m_i \geqslant 1)$

. Distinguons d'abord les différents groupements de n éléments parmi lesquels n_1 objets identiques, n_2 objets identiques a_2, ..., n_r objets identiques a_r, n_{r+1} objets différents $(\sum_{i=1}^{r+1} n_i = n)$ $(0 \leqslant n_i \leqslant m_i \ \forall i : 1..r+1)$

Cela peut se faire à l'aide de l'algorithme suivant :

```
pour i variant de 1 à r+1 répéter
    | n_i ←—— 0
grouper (n, r+1, m_1..m_{r+1})
```

```
programme grouper (a, b, c_1..c_{r+1})
    | si a = 0 alors
    |    | pour i variant de 1 à r+1 répéter
    |    |    | écrire n_i
    | sinon |
    |    | sommec ←—— 0
    |    | pour i variant de 1 à b répéter
    |    |    | sommec ←—— sommec + c_i
    |    | si sommec ⩾ a alors
    |    |    | pour i variant de 1 à b répéter
    |    |    |    | si a < c_i alors
    |    |    |    |    c_i ←—— a
    |    |    | n_b ←—— c_b
    |    |    | grouper (a-c_b, b-1, c_1..c_s)
    |    |    | si b > 1 alors
    |    |    |    |
    |    |    |    |
```

```
pour i variant de 1 à b répéter
    n_i ←—— 0
c_b ←—— c_b - 1
grouper (a, b, c)
```

Commentaires : D'abord, nous initialisons les n_i à 0 et exécutons le programme grouper avec comme paramètres le nombre d'éléments du groupement n, le nombre d'objets identiques +1, les nombres d'objets identiques et le nombre d'objets différents.
Grouper consiste à rechercer les groupements de a objets à choisir parmi les b classes d'objets, chacune de cardinal c_i (i:1..b).
Si a = 0, nous avons épuisé les objets et nous pouvons écrire le groupement de n objets.
Sinon, si la somme des éléments des classes ne suffit pas à constituer un groupement de a objets, c'est terminé.

sinon, nous utilisons deux fois le programme grouper récursivement.

1. nous garnissons la $b^{ème}$ classe du groupement de n objets à l'aide du minimum de (a,c_b) (si c_b est supérieur à a, nous avons plus d'éléments que nécessaire); ensuite, nous recherchons les groupements des $a-c_b$ éléments restants à l'aide des b-1 classes restantes
2. puisque nous avons recherché les groupements contenant min (a,c_b) éléments de la classe b, nous allons maintenant rechercher les groupements contenant min $(a,c_b)-1$ éléments de la classe b et cela uniquement dans le cas ou $b > 1$. Si b=1 nous devrions

former un groupement avec a éléments avec min $(a,c_b)-1$ éléments, ce qui est impossible.

. Pour chaque groupement, examinons combien il y a de cas possibles. Si tous les objets étaient différents, il y aurait n! façons de placer ces n objets dans n cases choisies. Cependant, les n_1 objets identiques a_1 peuvent être permutés sans donner de cas supplémentaires, de même pour $a_2,\ldots, a_r$, ce qui donne

$$\frac{n!}{n_1!\ n_2!\ \ldots\ n_r!}$$

. Enfin, il reste à choisir n cases parmi p sans tenir compte de l'ordre dans lequel ce choix est effectué.

Cela peut se faire de $\frac{p!}{(p-n)!\ n!}$ façons possibles.

. Nous avons donc au total

$$\frac{p!}{(p-n)!\ n!} \cdot \sum_{i=1}^{\text{nombre de groupements}} \frac{n!}{n_1!\ldots n_r!}$$

Note : lorsque m = p = n, nous obtenons la formule des permutations avec répétition, ce qui confirme notre traduction de la p. 35.

2ème partie.

Les trois outils présentés ci-dessous sont incompatibles comme décrit p.58. Ils feront donc chacun l'objet d'une étude particulière.

1. partitions.

Une action consiste à choisir n objets parmi m et à les placer dans r cases choisies parmi p. ($n \leqslant m$; $r \leqslant p$; $n \geqslant r$). Les objets doivent être tous différents et ne peuvent être répétés. Les cases peuvent être soit totalement distingables, soit indistingables. (comme précisé p.58).

a. cases totalement distingables.

. Choisissons d'abord n objets parmi m sans tenir compte de l'ordre dans lequel ce choix est effectué. Cela peut se faire de $\frac{m!}{(m-n)!\ n!}$ façons possibles.

. Choisissons ensuite r cases parmi p sans tenir compte de l'ordre dans lequel ce choix est effectué. Cela peut se faire de $\frac{p!}{(p-r)!\ r!}$ façons possibles.

. Enfin, disposant de n objets et r cases, nous devons placer les uns dans les autres. Si nous avions n cases, il y aurait n! façons de placer ces n objets dans ces m cases choisies. Cependant permuter n_1 objets se trouvant dans la même case ne donne pas de cas supplémentaire, de même pour toutes les cases contenant n_1 objets (au total r_1), de même pour les r_2 cases contenant n_2 objets, ..., de même pour les r_s cases contenant n_s objets, ce qui donne $\frac{n!}{(n_1!)^{r_1}\ (n_2!)^{r_2}\ldots(n_s!)^{r_s}}$.

. Nous avons donc au total

$$\frac{m!}{(m-n)!\ n!} \times \frac{p!}{(p-r)!\ r!} \times \frac{n!}{(n_1!)^{r_1}\ (n_2!)^{r_2}\ldots(n_s!)^{r_s}}$$

b. cases indistingables

La remarque de la p.60 concernant les cases indistingables est toujours d'application.

. Nous devons donc choisir n objets, ce qui donne $\frac{m!}{(m-n)!\ n!}$

. Pour placer les objets dans les cases, nous devons tenir compte du fait que permuter les objets entre les r_i cases contenant le même nombre d'objets ne donne pas de cas supplémentaire, ce qui donne

$$\frac{n!}{(n_1!)^{r_1}\ (n_2!)^{r_2}\ldots(n_s!)^{r_s} \times (r_1!\ r_2!\ldots r_s!)}$$

. Nous avons donc au total

$$\frac{m!}{(m-n)!\ n!} \times \frac{n!}{(n_1!)^{r_1}\ (n_2!)^{r_2} \ldots (n_s!)^{r_s} \times (r_1!\ r_2!\ \ldots r_s!)}$$

2. type d'action "répartir"

Une action consiste à choisir n objets parmi m et à les répartir dans r cases choisies parmi p. Comme précisé p.58, les objets doivent être tous différents et ne peuvent être répétés et les cases doivent être totalement distingables. Les deux prémières parties du raisonnement sont semblables à celles de 1. ci-dessus.

. Nous devons maintenant placer les n objets choisis dans les r cases choisies. Chacun de ces r objets peut être placé dans une des n cases. Cela peut se faire de r^n façons possibles.

. Nous avons donc au total

$$\frac{m!}{(m-n)!\ n!} \times \frac{p!}{(p-r)!\ r!} \times r^n$$

3. précision

Comme précisé p.58, une action consiste à choisir n objets parmi m et à les placer dans n cases choisies parmi p. Les objets doivent être tous différents et ne peuvent être répétés. Les cases doivent être indistingables. De plus, l'on ne peut mélanger les nombres maximum et minimum dans une même action.
Supposons que l'action est écrite comme suit :
placer n de %E avec n_1 de $\%E_1$ et n_2 de $\%E_2$ et ... et n_s de $\%E_s$ dans ... ($\#E = m$ et $\#E_i = m_i$, $n_i \leq m_i$ pour i : 1..s)
a. Supposons qu'il n'y ait pas de + derrière les n_i

alors $\sum_{i=1}^{s} n_i \leqslant n$.

Rénumérotons les indices de 1 à s de telle sorte que les n_q premiers nombres représentent le nombre exact d'éléments à choisir et que les autres représentent le minimum d'éléments à choisir. ($q \leqslant s$)
Nous pourrons donc avoir plusieurs groupements possibles de m éléments parmi lesquels t_i éléments de E_i ($1 \leqslant i \leqslant s$) et t_{s+1} éléments dans $E \setminus (\bigcup_{i=1}^{s} E_s)$.

$t_i = n_i$ pour $0 \leqslant i \leqslant q$

$0 \leqslant t_{s+1} \leqslant m - \sum_{i=1}^{s} m_i$

Soit $r_i = t_i - n_i$ pour $q + 1 \leqslant i \leqslant s$

$r_{s+1} = t_{s+1}$

Nous pourrons trouver les valeurs des r_i à l'aide de l'algorithme de la p.63 en recherchant les groupements de $n - \sum_{i=q+1}^{s} n_i$ éléments (c'est-à-dire le reste des éléments à choisir) parmi s-q+1 classes. Le cardinal de la première classe = $m_{q+1} - n_{q+1}$, ..., de la (s-q) ème classe = $m_s - n_s$, de la (s-q+1) ème classe = $m - \sum_{i=1}^{s} n_i$.

b. Supposons qu'il y ait au moins un + derrière un m_i.

Rénumérotons les indices de 1 à s de telle sorte que les n_q premiers nombres représentent le nombre exact d'éléments à choisir et que les autres représentent le maximum d'éléments à choisir ($q \leqslant s$).
Nous pourrons donc avoir plusieurs groupements possibles de n éléments parmi lesquels t_i éléments de E_i ($1 \leqslant i \leqslant s$) et t_{s+1} éléments dans

$E \setminus (\bigcup_{i=1}^{s} E_s)$

$t_i = n_i$ pour $0 \leqslant i \leqslant q$

$t_i \leqslant n_i$ pour $q+1 \leqslant i \leqslant s$

$0 \leqslant t_{s+1} \leqslant m - \sum_{i=1}^{q} m_i$

Nous pourrons trouver les valeurs des t_i à l'aide de l'algorithme de la p.63 en recherchant les groupements de $n - \sum_{i=1}^{q} n_i$ éléments (c'est-à-dire le reste des éléments à choisir) parmi s-q+1 classe. Le cardinal de la première classe = n_{q+1} (maximum des éléments que l'on peut prendre), de la (s-q) ème classe = n_s, de la (s-q+1) ème classe = $m - \sum_{i=1}^{q} m_i$.

. Pour chaque groupement ainsi trouvé, nous devons choisir t_1 objets parmi m_1, t_2 objets parmi m_2, ..., t_s objets parmi m_s, t_{s+1} parmi $m - \sum_{i=1}^{s} m_i$ sans tenir compte de l'ordre dans lequel ce choix est effectué.
Cela peut se faire de

$$\frac{m_1!}{(m_1-t_1)!\ t_1!} \times \ldots \times \frac{m_s!}{(m_s-t_s)!\ t_s!} \times \frac{(m - \sum_{i=1}^{s} n_i)!}{(m-\sum_{i=1}^{s} m_i-t_{s+1})!\ t_{s+1}}$$

façons possibles.

. Enfin, il reste à choisir n cases parmi p sans tenir compte de l'ordre dans lequel ce choix est effectué.

Cela peut se faire de $\frac{p!}{(p-n)!\ n!}$ façons possibles.

. Nous avons donc au total $\frac{p!}{(p-n)!\ n!} \cdot \sum_{i=1}^{\text{nombre de groupements}}$

$$\frac{m_1!}{(m_1-t_1)!\ t_1!} \times \ldots \times \frac{m_s!}{(m_s-t_s)!\ t_s!} \times \frac{(m-\sum_{i=1}^{s} m_i)!}{(m-\sum_{i=1}^{s} m_i-t_{s+1})!\ t_{s+1}!}$$

Chapitre 3. Analyse fonctionnelle du système.

3.1. Schéma global du système.

Nous allons donc tenter de distinguer les différentes procédures constituant le système pour ensuite les décrire plus précisément. Pour ce faire, nous simplifierons le problème en ne tenant compte que d'un seul exercice et en oubliant le décalage de temps entre l'utilisation du système par l'enseignant et l'élève. La première hypothèse n'est pas très restrictive : pour être complet, nous devons uniquement fournir un outil capable de gérer la séquence des exercices qui ne fera pas partie du présent travail et reste à développer. La seconde hypothèse nous amène à laisser de côté momentanément le problème de ce qui devra être mémorisé dans le système. De plus, cette analyse ne concernera pas l'utilisation du langage par l'élève qui sera, elle aussi, susceptible de développements futurs.

D'abord, le système sera considéré comme un tout et nous étudierons ses différentes fonctions. Celles-ci serviront ensuite à définir des procédures et nous étudierons les différents messages, qu'ils soient internes au système, en provenance ou à destination de l'élève ou de l'enseignant.

L'enseignant introduit donc deux énoncés, l'un en langage naturel (profeln), l'autre en langage formalisé (profelf). Ensuite le système présente l'énoncé en langage naturel (eleln) à l'élève qui fournit une réponse (elrep) à cet exercice. Enfin, en fonction de celle-ci, des explications sont présentées à l'élève (elexpl.) (fig. 1)

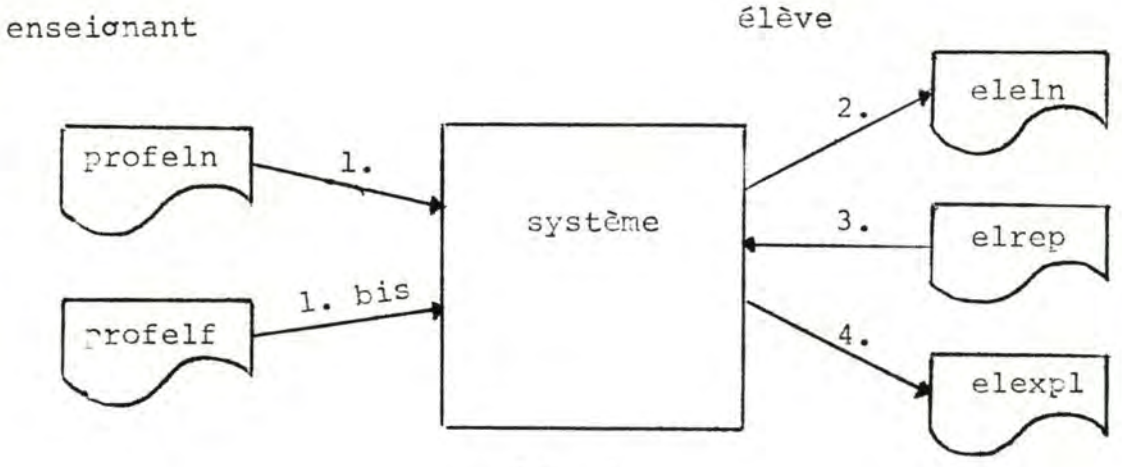
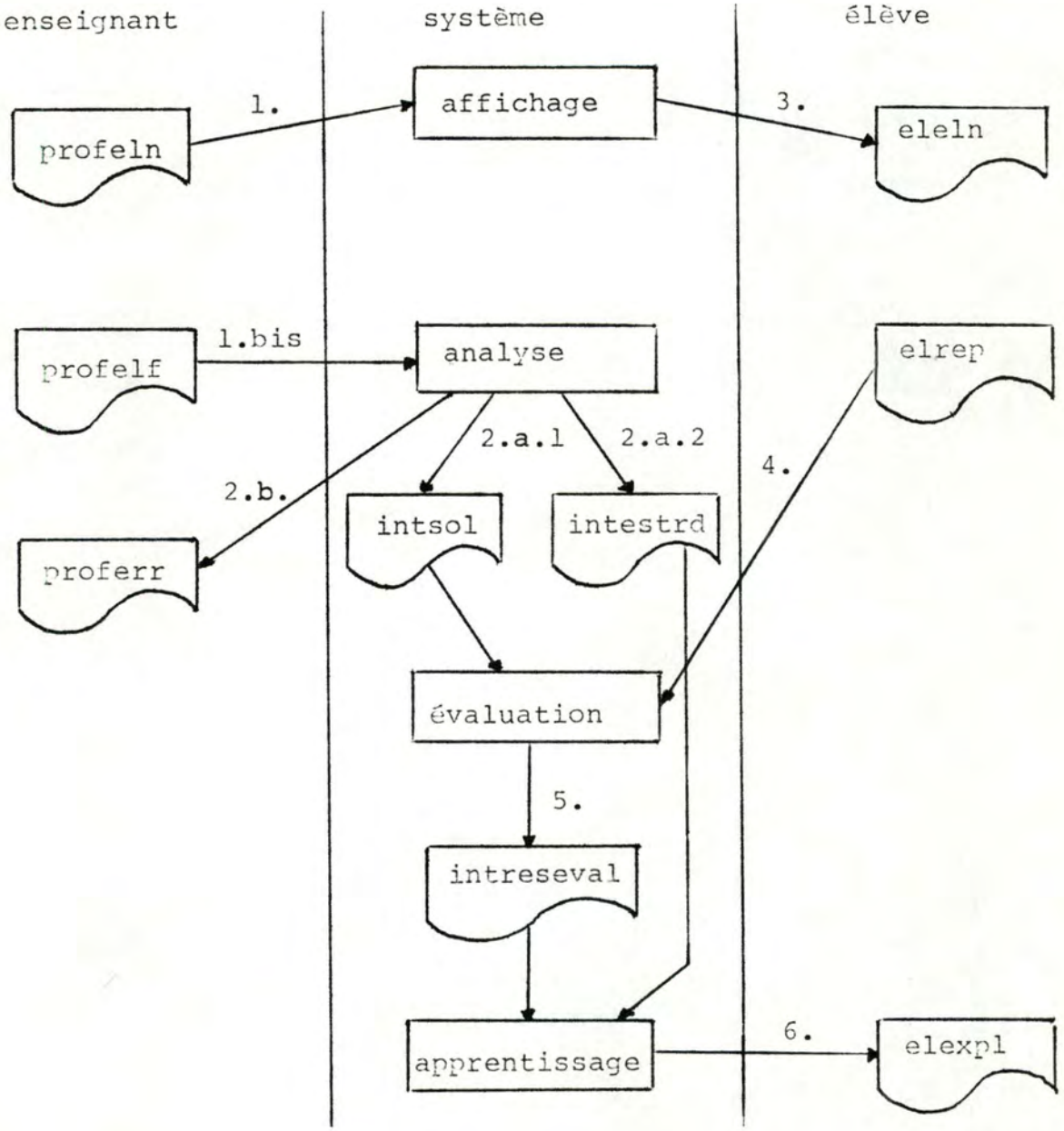


fig. 1

Nous pouvons distinguer une première procédure très simple du système : elle consiste à afficher à l'écran l'énoncé en langage naturel. En ce qui concerne le second énoncé, nous avons supposé ci-dessus qu'il était correct au vu de la définition du langage du deuxième chapitre, ce qui ne sera pas toujours le cas. C'est pourquoi, nous devons prévoir une procédure d'analyse qui si l'énoncé n'est pas correct, présente au professeur les erreurs qu'il a faites en rédigeant l'énoncé (proferr). Ensuite la réponse donnée par l'élève doit être comparée à la solution de l'exercice (intsol) calculée à partir de l'énoncé en langage formalisé ou plus précisément, nous profiterons de l'analyse de l'énoncé pour calculer la solution qui exige de toutes façons elle aussi, une analyse. Les solutions seront comparées dans une procédure d'évaluation. Enfin, en fonction du résultat de cette évaluation (intreseval) et toujours sur base de l'énoncé en langage formalisé, une dernière procédure d'apprentissage fournira les explications adéquates à l'élève. Ici aussi, nous profiterons de l'analyse pour créer une structure de données représentant l'énoncé en langage formalisé d'une manière beaucoup plus aisément utilisable par la suite (intestrd), lors des explications. (fig. 2)

La chronologie décrite ci-dessus devrait mentionner les boucles suivantes : d'abord, il y a un va et vient entre le professeur et le système tant que l'énoncé n'est pas correct. Ensuite, tant que le système n'est pas satisfait de

l'élève, il lui repose des questions.



3.2. Définition des traitements.

Nous allons maintenant décrire brièvement les 4 procédures suivant le modèle suivant : nom de la procédure, messages en entrées, messages en sorties, fonction.

1. affichage

reçoit : profeln
produit : eleln
fonction : afficher pour l'élève l'énoncé en langage naturel introduit par l'enseignant.

2. analyse

reçoit : profelf
produit : intsol, intestrd, proferr
fonction : si profelf est correct (cf définition du langage au chapitre 2.), alors,
. traduire profelf en intestrd plus aisément manipulable.
. calculer la solution de l'exercice intsol
sinon,
. présenter les erreurs proferr

3. évaluation

reçoit : intsol, elrep
produit : intreseval
fonction : comparer intsol et elrep et placer dans intreseval le résultat de cette comparaison

4. apprentissage

reçoit : intreseval, intestrd
produit : elexpl
fonction : en fonction de intreseval et sur base de intestrd, fournir les explications correspondantes à l'élève elexpl

3.3. Définition des données.

PROFELN - ELELN : ce sont simplement l'énoncé en langage naturel introduit par l'enseignant et cet énoncé présenté à l'élève.

PROFELF : énoncé traduit dans le langage formalisé défini au chapitre 2.

PROFERR : numéro d'erreur, numéro de ligne et numéro de caractère

où elle a été décelée.

Le numéro d'erreur renvoie à la liste d'erreurs décrite en annexe 2. Etant donné que la description d'un exercice dans le langage est tout de même assez brêve et par conséquent contient peu ou pas d'erreurs, nous avons choisi de nous arrêter à la première erreur rencontrée. Ce choix simplifie l'analyse : le programme analysant le langage ne doit pas tenter de récupérer les erreurs pour comprendre la suite de l'exercice.

INTSOL : description de la solution de l'exercice la plus précise possible.

Pour chaque action, les différentes parties du raisonnement présentées en 2.3. seront mémorisées. Cela permettra ensuite de déceler plus précisément où se trouvent les erreurs éventuelles dans le raisonnment de l'élève. Si nous avions uniquement décidé de mémoriser la solution finale de l'exercice, nous devrions, chaque fois qu'il y a erreur dans le raisonnement, expliquer toute la solution à l'élève même s'il y a uniquement une infime partie qu'il n'a pas compris.

INTESTRD : structure de données représentant l'énoncé en langage formalisé.

Nous devons donc représenter les différents concepts qu'un énoncé est susceptible d'utiliser. Pour ce faire, nous allons employer l'approche ENTITE/ASSOCIATION [8] qui est le modèle de représentation de données le plus répandu.

D'abord, le modèle entité/association de l'exercice sera représenté graphiquement à la fig. 3. Ensuite nous définirons les entités, les associations et les propriétés. Une entité sera définie par son nom, éventuellement un synonyme, sa description, ses identifiants et propriétés éventuelles. Une association sera définie par son nom, éventuellement un synonyme, sa description, ses propriétés éventuelles, les entités qu'elle associe et enfin la connectivité. Un groupe

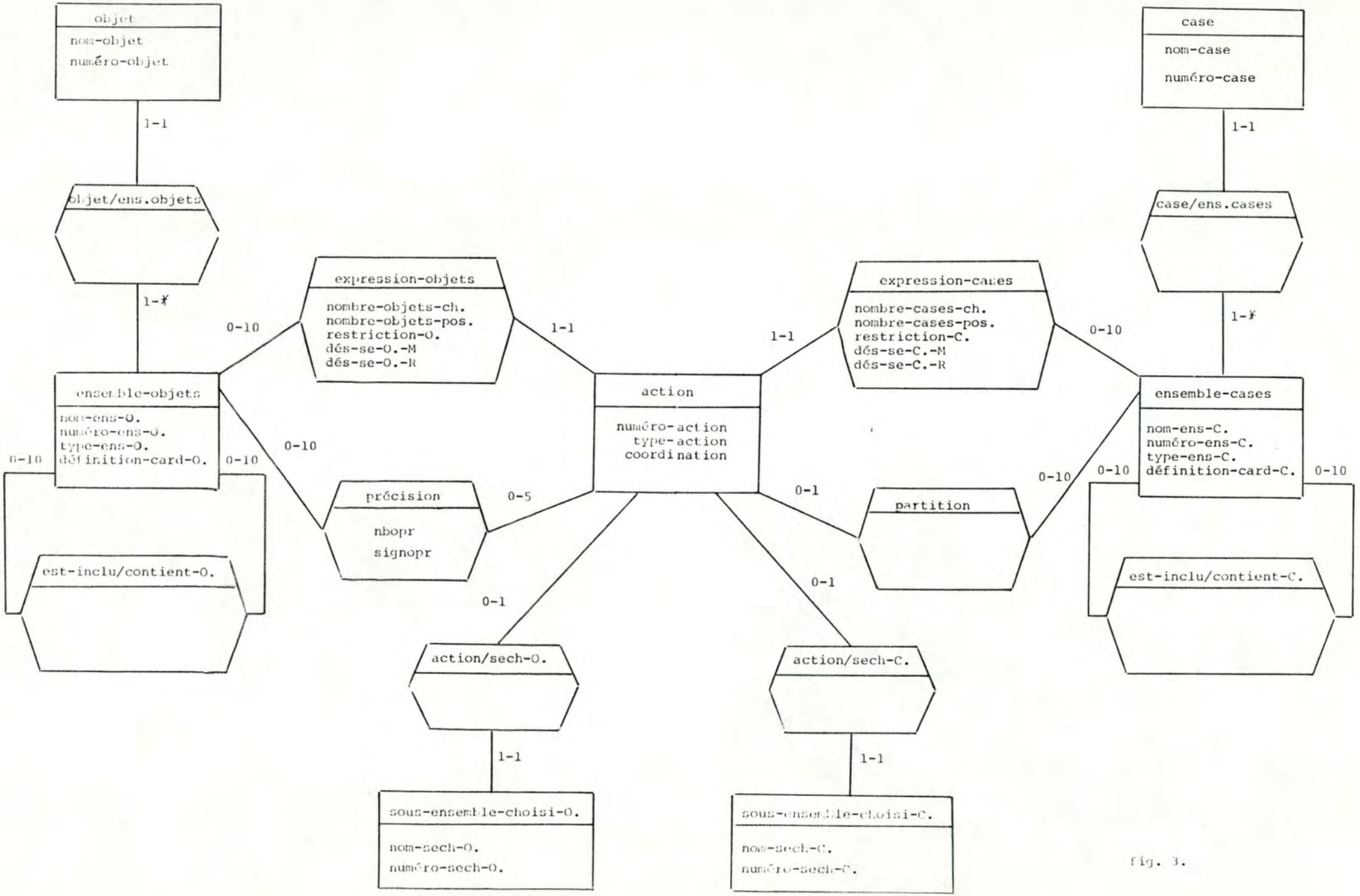

objet
nom-objet
numéro-objet
1-1
objet/ens.objets
1-*
ensemble-objets
nom-ens-O.
numéro-ens-O.
type-ens-O.
définition-card-O.
0-10
0-10
0-10
0-10
est-inclu/contient-O.
expression-objets
nombre-objets-ch.
nombre-objets-pos.
restriction-O.
dés-se-O.-M
dés-se-O.-R
1-1
précision
nbopr
signopr
0-5
action
numéro-action
type-action
coordination
0-1
0-1
action/sech-O.
1-1
sous-ensemble-choisi-O.
nom-sech-O.
numéro-sech-O.
0-1
action/sech-C.
1-1
sous-ensemble-choisi-C.
nom-sech-C.
numéro-sech-C.
1-1
expression-cases
nombre-cases-ch.
nombre-cases-pos.
restriction-C.
dés-se-C.-M
dés-se-C.-R
0-10
0-1
partition
0-10
case
nom-case
numéro-case
1-1
case/ens.cases
1-*
ensemble-cases
nom-ens-C.
numéro-ens-C.
type-ens-C.
définition-card-C.
0-10
0-10
est-inclu/contient-C.

fig. 3.

de propriétés sera défini par son nom, éventuellement un synonyme, sa description et les propriétés qu'elle contient. Une propriété sera définie par son nom, éventuellement un synonyme, son format et sa description. La description ne sera donnée que lorsque le nom renvoyant à la définition du langage du chapitre 2 n'est pas suffisamment explicite. les contraintes d'intégrité seront inclues aux définitions. Ces définitions figurent en annexe 4. Notons qu'elles ne tiendront pas compte de la possibilité qui est offerte d'utiliser des variables. (cf p.49)

ELREP : réponse de l'élève.

L'élève sera invité à décrire sa réponse à l'aide d'une expression, ce qui permettra; si elle contient une erreur, de déceler plus précisément où elle se trouve. Il pourra cependant donner uniquement la réponse finale s'il le désire.

INTRESEVAL : suite de numéros d'action-numéros de parties du raisonnement.

Intreseval signalera ainsi quelles parties du raisonnement sont exactes.

ELEXPL : présentation des explications à l'élève.

La procédure apprentissage ne sera pas développée dans ce travail. Par conséquent, nous donnerons uniquement quelques lignes directrices pour définir elexpl. Nous allons présenter brièvement les explications fournies à l'élève au cas où intreseval ne comprend rien, c'est-à-dire dans le cas où nous ne pouvons distinguer aucune ressemblance entre elrep et intsol.

D'abord, nous expliquerons la méthode de résolution en paraphasant les différentes actions du raisonnement. Voici quelques exemples de ce que l'on pourrait trouver.

ex. 1 (vocabulaire : cf. annexe 4)
si type-ens-O. de l'ensemble d'objets utilisé dans l'action en question = ossrep
et si type-ens-C. de l'ensemble d'objets utilisé dans l'action en question ≠ bssord
et si nboch = nbcch = n, nbop = m, nbcp = p (m, n, p ⩾ 1),
on écrira la phrase suivante :
choisir n (nom-d'ensemble-d'objets) différent(e)s parmi m et placer dans n (nom-d'ensemble-de-cases) différent(e)s choisi(e)s parmi p.

ex. 2 exercice n° 7
choisir 3 valeurs d'un dé parmi 6.

ex. 3 exercice n° 13
choisir 1 ami (qu'on appelle x) parmi 2 et placer dans 1 place (qu'on appelle i) choisi(e) parmi n-r-1 compris(es) entre $1^{er(e)}$ et $(n-r-1)^{ème}$.
et prendre 1 ami parmi 2 - 1 (à savoir x) et placer dans 1 place (qu'on appelle j) à savoir $i+r+1^{ème}$
et choisir n-2 autres personnes parmi n-2 et placer dans n places - 2 (à savoir p et q)

ex. 4 exercice n° 9 (seconde traduction)
choisir 6 pavillons parmi 6 dont 2 sont des pavillons-rouges, 2 sont des pavillons-verts, 2 sont des autres-pavillons.

ex. 5 exercice n° 19 (seconde traduction)
choisir au moins 1 fille parmi 3 pour constituer un ensemble de 4 élèves choisis différents.

Ensuite pour chaque action, nous demanderons une réponse à l'élève. Si celle-ci n'est pas bonne, nous donnerons des explications plus précises, notamment en décomposant cette action en plusieurs sous-actions quand cela est possible (ex : exercice n° 19 première traduction) ou alors en rensei-

gant l'élève sur le type d'objets et le type de cases. Nous progresserons de cette manière tant que l'élève ne donnera pas une réponse suffisante ou que les explications possibles seront épuisées.

Enfin, nous demanderons à nouveau une réponse à l'ensemble de l'exercice.

Chapitre 4. Analyse organique du système.

Levons maintenant l'hypothèse selon laquelle il n'y a pas de décalage de temps entre l'utilisation du système par l'enseignant et l'utilisation par l'élève. Dans un premier temps, l'enseignant introduit dans la machine l'énoncé en langage naturel et l'énoncé en langage formalisé. La procédure d'analyse doit alors lui répondre si son énoncé est correct ou non. Dans un second temps, l'élève introduit sa solution. En fonction du résultat de la comparaison de celle-ci à celle calculée à partir de l'énoncé en langage formalisé, la procédure d'apprentissage génère les explications.

Nous sommes placés devant une alternative en ce qui concerne intsol et intestrd. Soit nous conservons uniquement en mémoire auxiliaire les énoncés en langage formalisé et en langage naturel et chaque fois qu'un élève doit résoudre un exercice, nous l'analysons de nouveau, soit nous mémorisons intsol et intestrd une fois profelf correct. Cette distinction correspond à celle que l'on fait entre l'interprétation et la compilation d'un langage de programmation. La première solution nous permet de gagner de la place en mémoire auxiliaire mais nécessite plus de temps. Nous devons donc choisir entre un gain de temps ou un gain de place en mémoire auxiliaire. Le temps est une ressource moins essentielle : après avoir présenté l'énoncé en langage naturel à l'élève, celui-ci doit réfléchir et nous pouvons profiter de ce temps de réflexion pour analyser l'énoncé en langage formalisé. Par contre, économiser la place en mémoire nous permet d'avoir une très grande banque d'exercices en permanence, ce qui nous paraît très avantageux. Nous avons par conséquent choisi d"interpréter" le langage à moins que le temps nécessaire pour ce faire ne soit vraiment trop important. Dans ce cas, nous choisirions l'autre solution.

L'analyse organique concernera uniquement la procédure d'analyse, les autres procédures restant à développer.

Extensions possibles.

Voyons maintenant quels pourraient être les développements ultérieurs de ce travail. Commençons évidemment par les extensions nécessaires pour obtenir un produit fini : nous devrons implémenter la procédure d'affichage (ce qui est évident), la procédure d'évaluation (qui ne devrait pas présenter beaucoup de difficultés si l'on décide de ne pas utiliser de nombres variables) et enfin la procédure d'apprentissage qui devrait représenter un travail assez conséquent. Il serait utile que celle-ci soit expérimentée dans une classe et retravaillée au vu des remarques des enseignants.

L'utilisation de variables nécessite également des extensions dont la difficulté essentielle consistera en la comparaison de la réponse du système à la réponse de l'élève. En effet, deux expressions peuvent avoir une forme différente et représenter en fait la même solution. Un exercice peut en effet être résolu de plusieurs façons différentes.

Ceci nous amène à une autre extension très importante : plutôt que de se limiter à un énoncé en langage formalisé, les enseignants auront la possibilité d'introduire plusieurs énoncés en langage formalisé par exercice, chacun représentant une façon de résoudre celui-ci et générant la même solution mais exprimée de façons différentes. La procédure d'évaluation comparera la solution fournie par l'élève aux différentes solutions en provenance des énoncés en langage formalisé. Si nous rencontrons une expression de solution qui "ressemble" (ce terme restant à définir) à celle de l'élève, il n'y a pas de problèmes. Par contre si l'expression de la solution ne peut être comparée à aucune autre, nous avons deux possibilités :

. soit la solution est correcte (ce qui peut être calculé) par un heureux hasard ou grâce à un raisonnement de l'élève auquel le professeur n'a pas pensé. Dans ce cas, il sera utile que le système mémorise cette solution pour que

le professeur, par un système de boîte aux lettres puisse examiner ce raisonnement nouveau qui, s'il est correct, peut l'amener à rédiger un nouvel énoncé en langage formalisé.
. soit elle est erronée et alors nous devons présenter à l'élève la solution que le professeur juge la plus adéquate.

Suite à ce qui a été annoncé p. 70, il sera aussi très utile de fournir un outil gérant la séquence des exercices en fonction des réponses données par les élèves. Ceci pose le problème du suivi qui est à lui seul assez vaste pour mériter une étude particulière.

Reprenons aussi la suggestion de l'utilisation du langage par l'élève. Cette idée pédagogiquement originale l'obligerait à un raisonnement très systématique et pourrait être de nature à améliorer sensiblement le résultat de celui-ci.

Enfin, le langage est lui aussi susceptible de modifications et extensions inspirées par maintes réflexions. La plus importante concerne les cases indistingables. Ce type d'ensemble a en fait été employé pour des cases de deux natures différentes : bien sûr pour des cases indistingables mais aussi pour des cases dont l'ordre n'a pas d'importance, ce qui est tout de même un peu différent. Ensuite, les exercices traduits au moyen du type d'action "répartir" pourraient être exprimés plus élégamment au moyen d'un nouveau type d'ensembles de cases : les cases à répétition par analogie aux objets et dans lesquelles on pourrait placer plusieurs objets. Enfin, des facilités d'écriture supplémentaires pourraient être offertes. Par exemple, lorsqu'un exercice comporte plusieurs parties, il serait utile de ne pas répéter les mêmes déclarations chaque fois mais de les écrire une seule fois, pour toutes les parties de l'exercice. Une autre facilité consisterait à ne pas devoir déclarer des ensembles dont la définition est évidente pour tous (exemple : chiffres, lettres, etc...).

Conclusion

En portant un jugement à court terme sur ce travail, le résultat pourrait paraître assez décevant. En effet, dans sa forme actuelle, le système constitue uniquement un outil d'aide à l'enseignement assisté par ordinateur de l'analyse combinatoire et n'est par conséquent pas utile aux enseignants, à moins que leur formation en informatique ne soit assez solide pour leur permettre de continuer ce qui a été commencé.

Pourtant, en examinant les différentes extensions présentées ci-dessus, nous pouvons légitimement espérer que ce travail soit à la base de plusieurs autres qui en démontreraient toute la portée.

En optant pour la seconde forme d'enseignement assisté par ordinateur (cf p. 9), nous avons donc fait un pari sur l'avenir en privilégiant un domaine où les découvertes furent nombreuses plutôt qu'un produit fini dont le développement est moins stimulant mais le résultat plus immédiat.

Nous croyons enfin que ce projet d'enseignement assisté par ordinateur qui consiste à "inculquer" à l'ordinateur des "connaissances" propres à une matière précise, limitée et bien définie peut servir de modèle à des projets semblables dans d'autres domaines.

Bibliographie

[1] DONNAY, J. : Variables changeables et variables statiques (texte provisoire), Facultés Notre-Dame de la Paix, Unité de pédagogie, 1983

[2] DE MONTMOLLIN, M. : L'enseignement programmé, Presses Universitaires de France (Que sais-je ?), Paris, 1975

[3] LIPSCHUTZ, S. (Temple University) : Probabilités. Cours et problèmes (Série Schaum), Ediscience S.A., 1973

[4] LOUQUET, T., TRIBOULEy, J., VOGT, A. : Probabilités (Les mathématiques en terminales C et E), Armand Collin

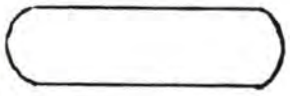
[5] BAUVIGNET, R., : Analyse combinatoire (Syllabus de première candidature en sciences économiques et sociales), Facultés Notre-Dame de la Paix Namur

[6] FOURASTIE, J. et SAHLER, D. : Probabilités et statistique (Série J. QUINET), Dunod, 1978

[7] COMBES, A. et SAADA, M. : Exercices et problèmes de probabilités et de statistique avec solutions (Classes de 1ère terminales), Vuibert

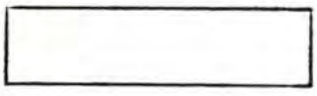
[8] CHEN, P.P. : The entity - relationship model. Toward a unified view of data, in ACM TODS, Vol. 1, n° 1, 1976

Annexe 1. Syntaxe du langage.



représente - un mot réservé du langage (ex : placer)
ou - une entité syntaxique non définie plus loin car sa définition est évidente pour tous (ex : lettres, chiffres, etc ...)

représente un opérateur du langage (ex : ;)



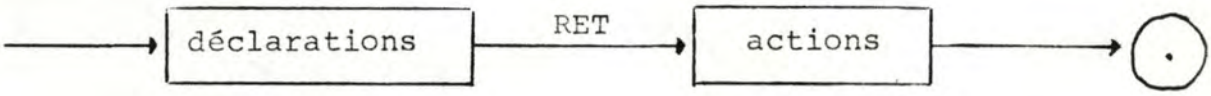
représente une entité syntaxique pour un autre diagramme

Nous commençons par définir un exercice comme des déclarations suivies d'actions, le tout terminé par un point. Nous définissons ensuite les déclarations et les actions et ainsi de suite jusqu'à ce que toute entité syntaxique complexe soit définie et qu'il ne reste plus que des opérateurs, des mots réservés ou des entités syntaxiques dont la définition est évidente pour tous. Ajoutons que b indique qu'à cet endroit, il faut laisser un espace et que RET indique qu'il faut écrire une fin de ligne.

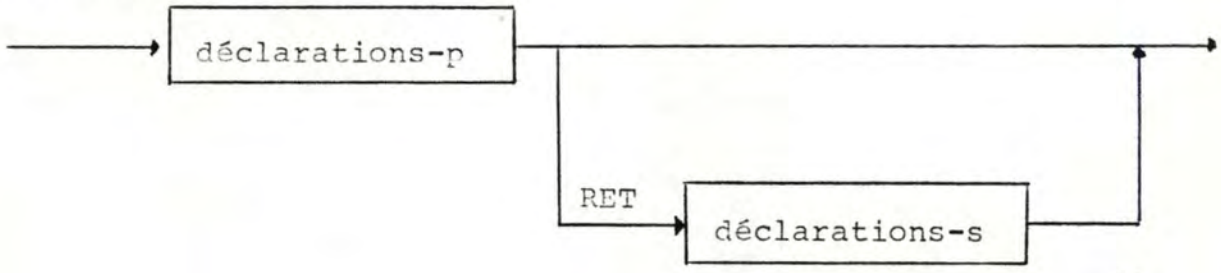
Voyons par exemple comment sont définies les déclarations primaires (déclarations-p). Nous devons d'abord écrire un nom d'ensemble (qui est défini plus loin) suivi d'un espace. Nous rencontrons alors une alternative correspondant au choix qui nous est offert de citer les éléments ou non. Si nous ne les citons pas, nous devons écrire le symbole "#" suivi d'un espace, lui-même suivi d'un nombre (qui est défini plus loin). Ensuite nous nous trouvons devant un nouveau choix correspondant à la description du type de l'ensemble. Nous devons écrire un des six symboles suivi d'un ";" ou directement un "." si l'on choisit la dernière branche. Enfin,

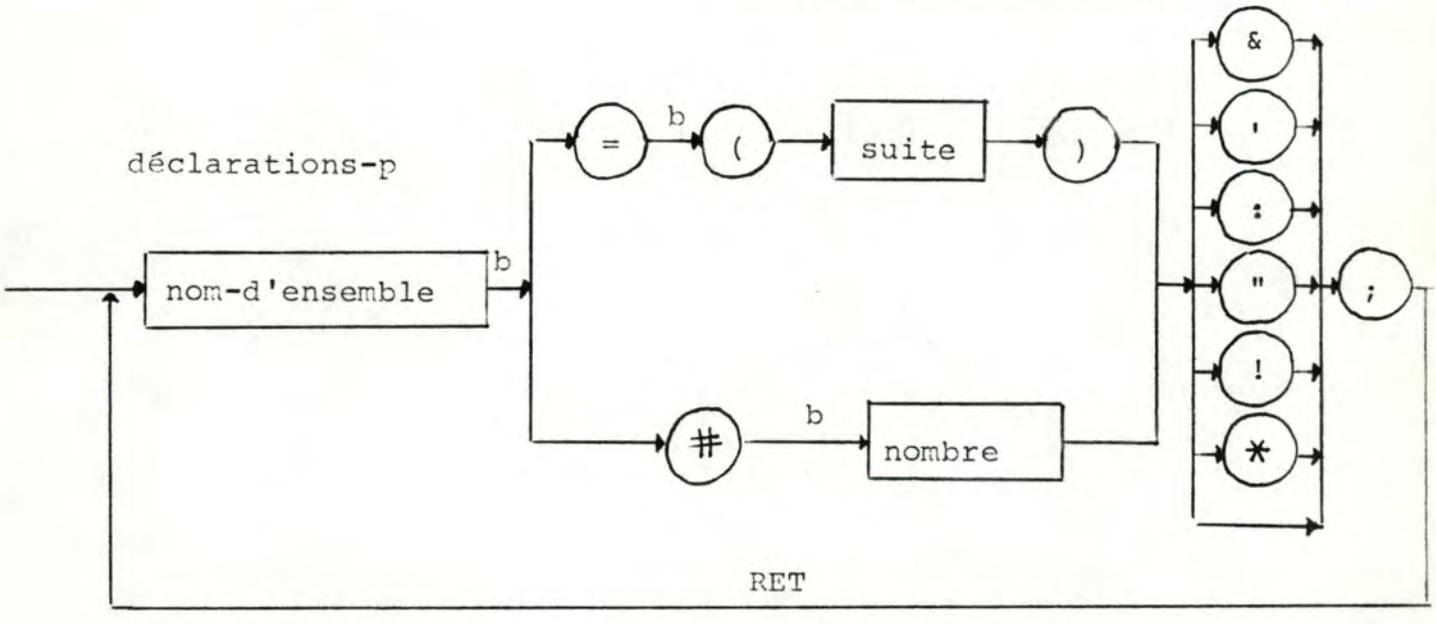
nous rencontrons une dernière alternative : soit terminer les déclarations primaires, soit écrire une fin de ligne et écrire une nouvelle ligne de déclarations primaires.

exercice

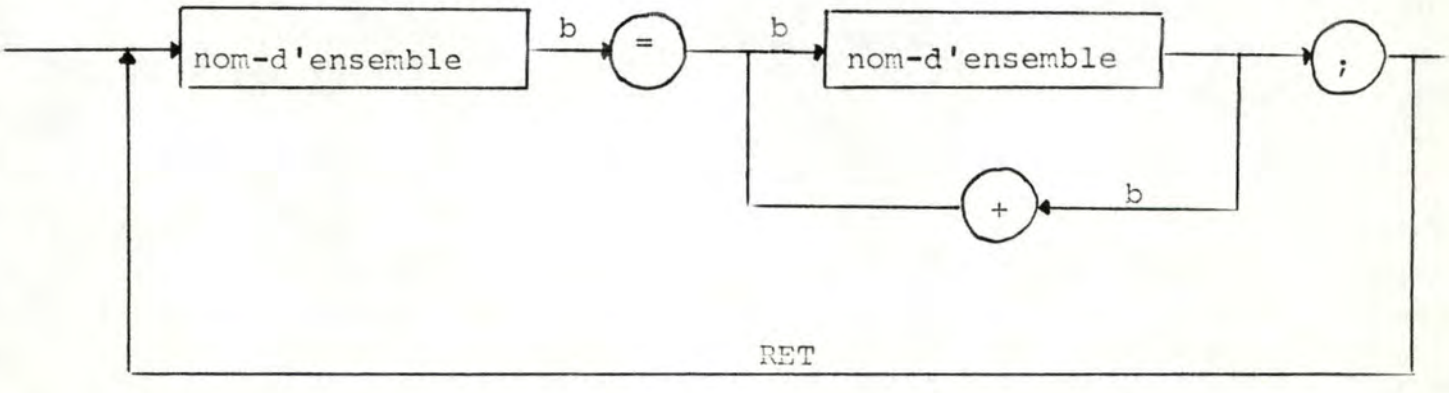


déclarations

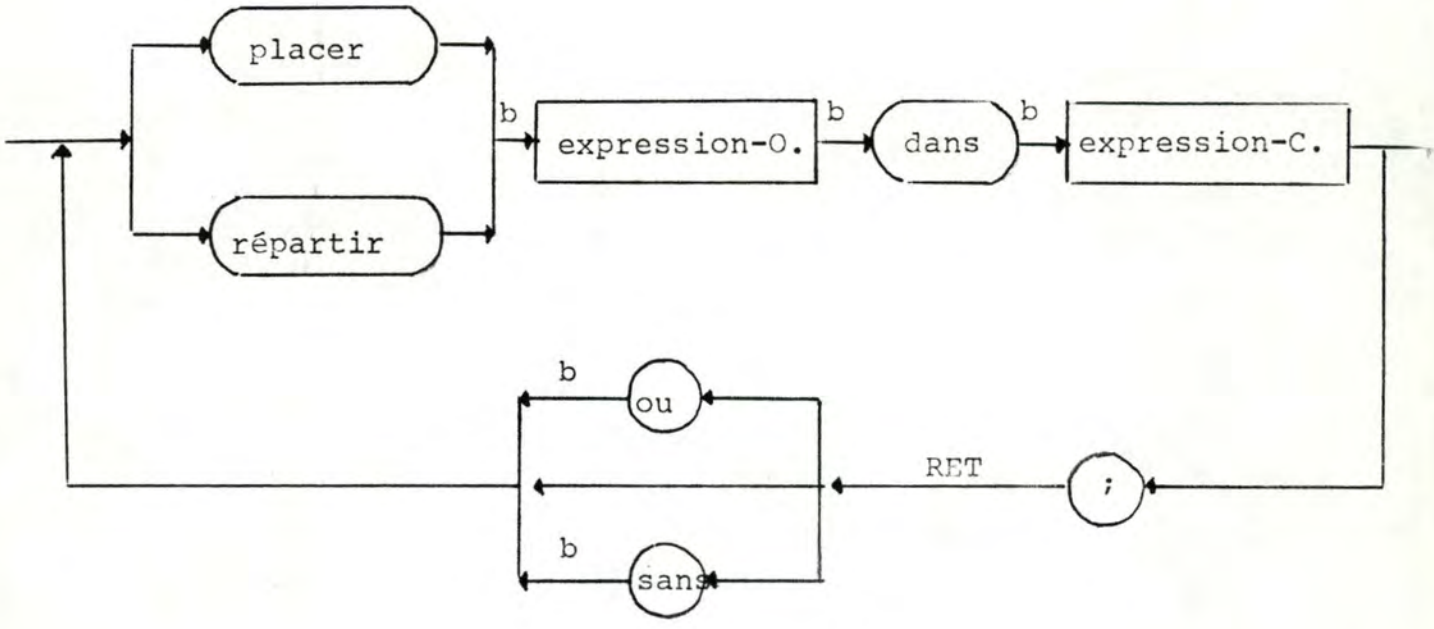




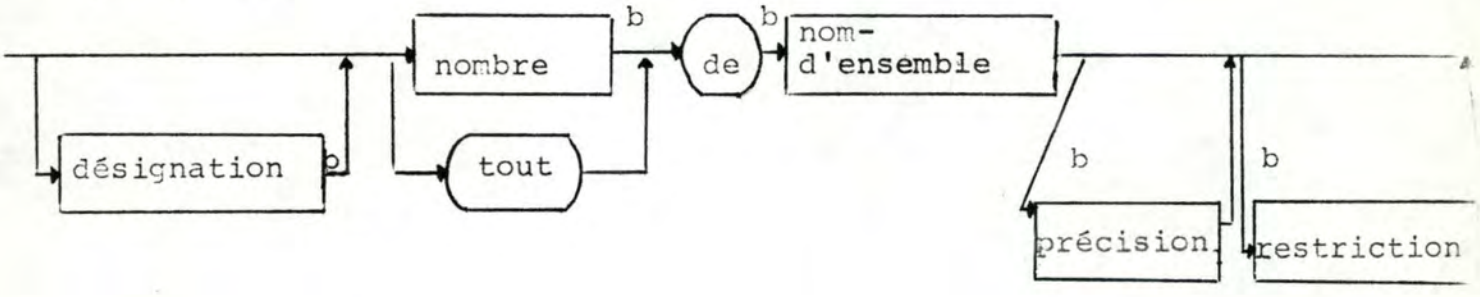
déclarations-s



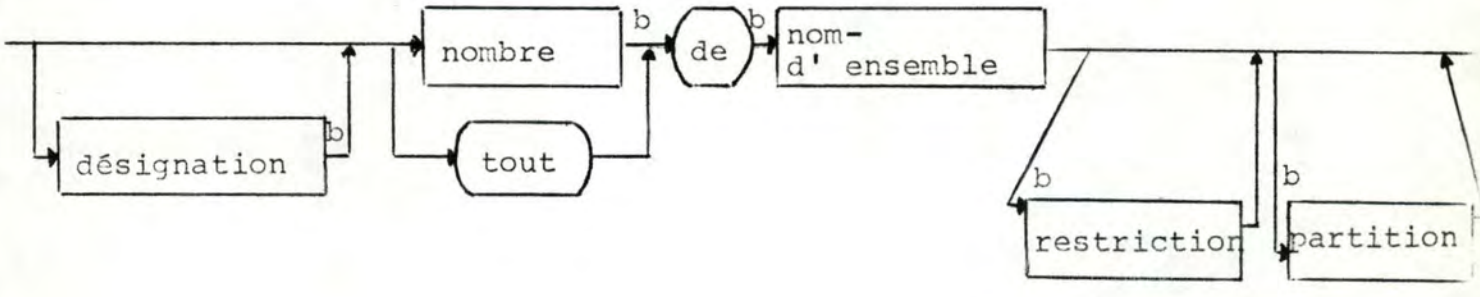
actions



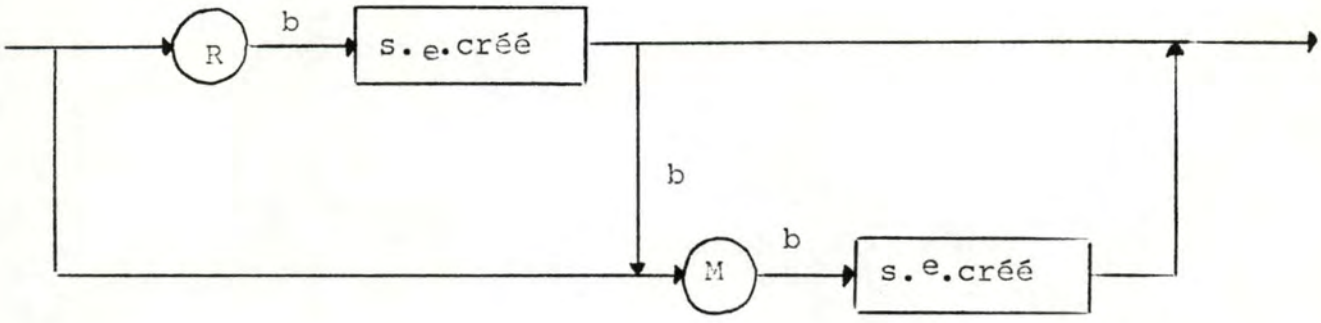
expression-O.



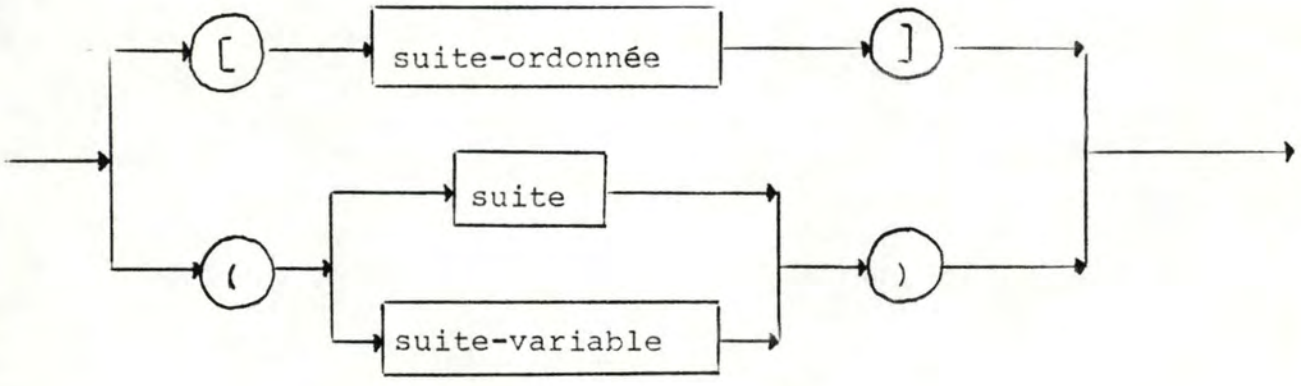
expression-C.



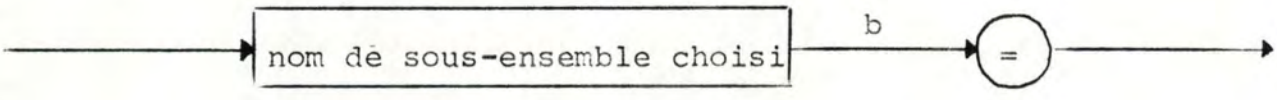
restriction



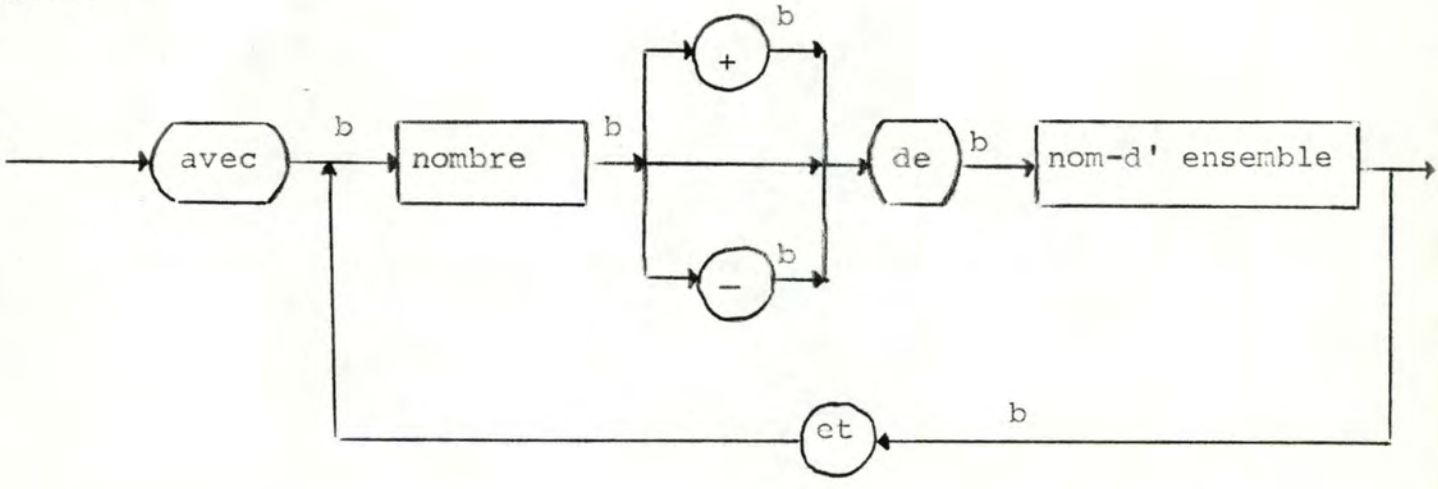
s.e.créé



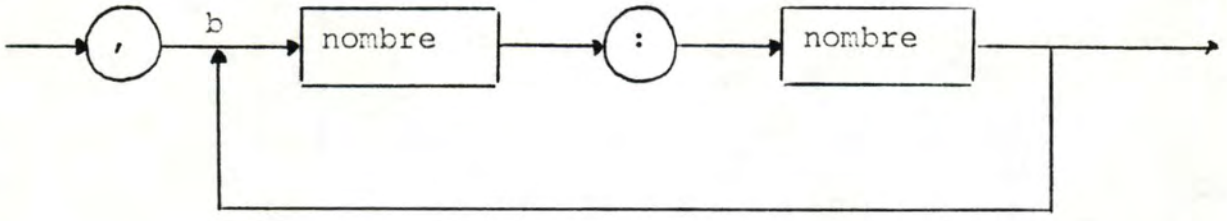
désignation



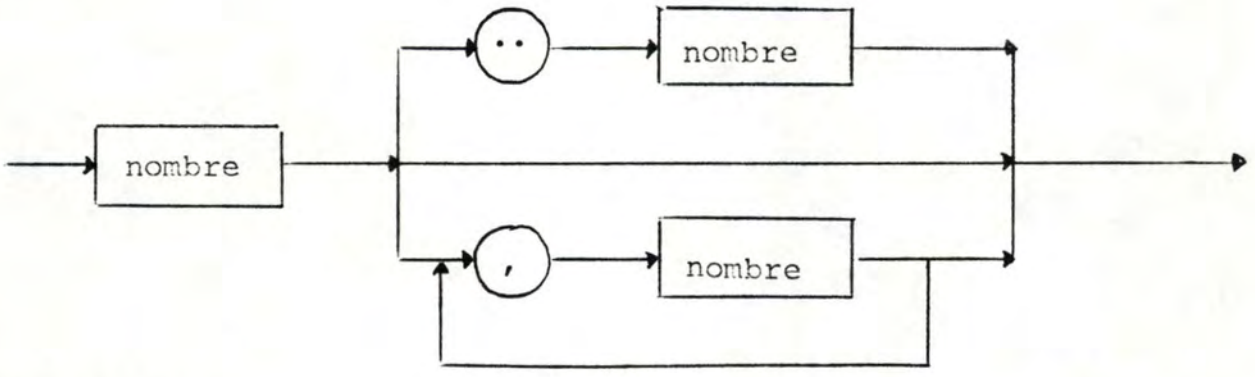
précision



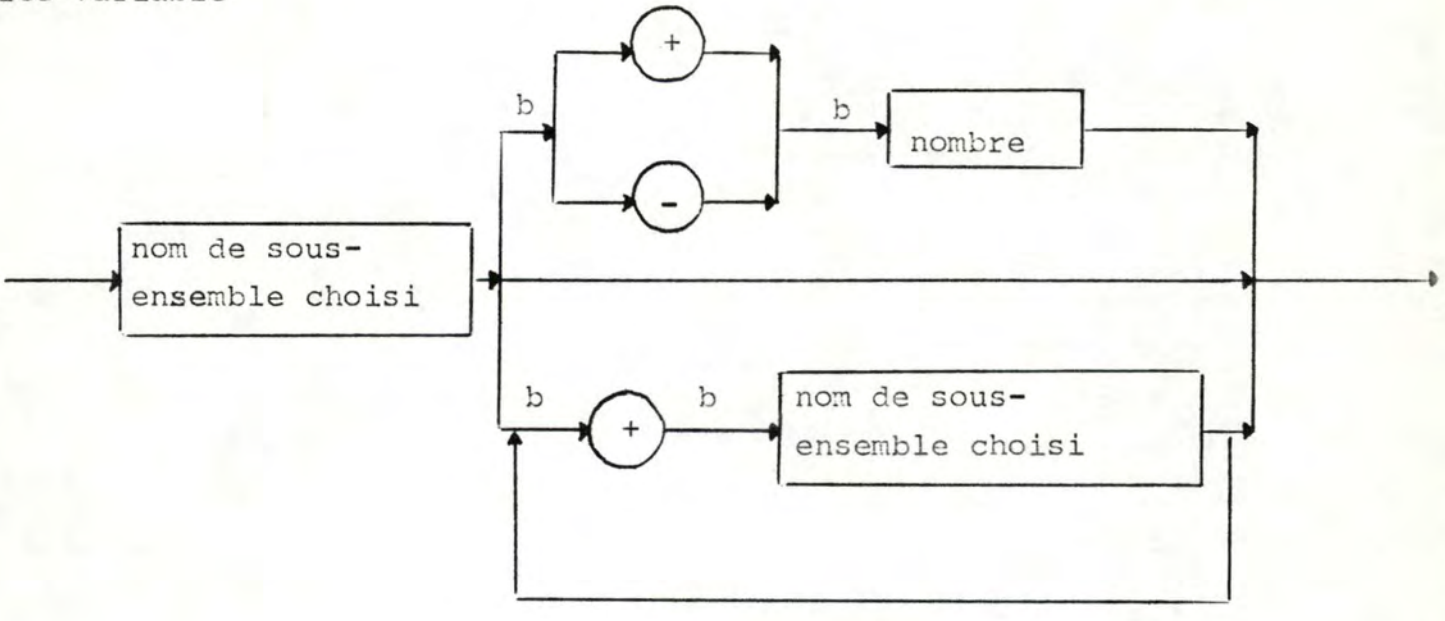
partition



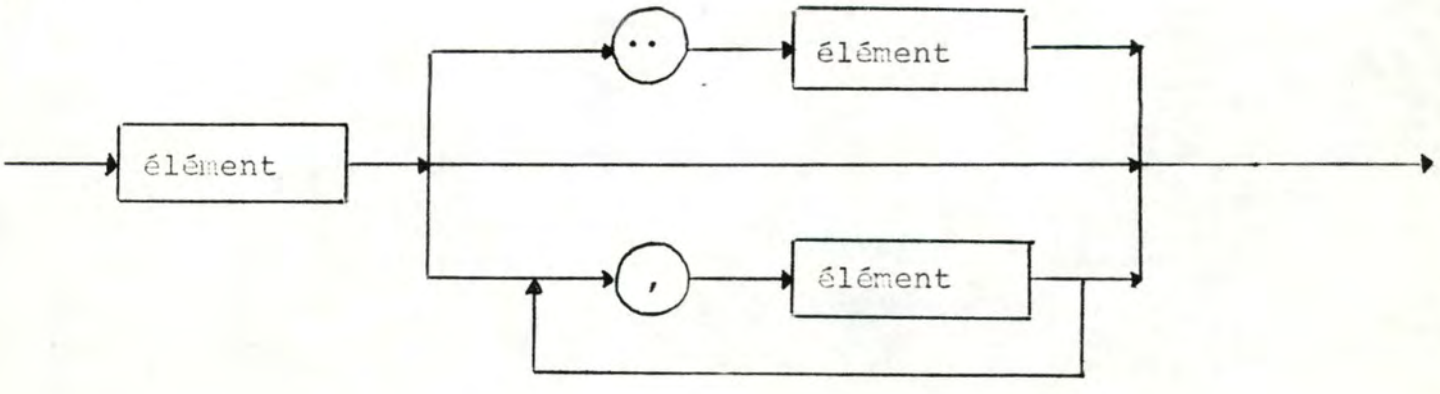
suite-ordonnée



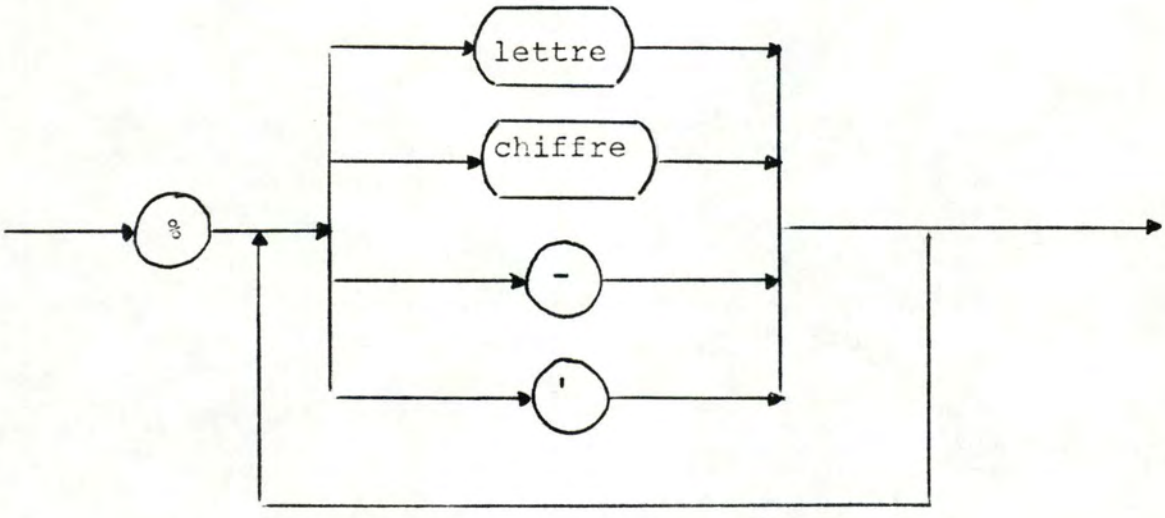
suite-variable



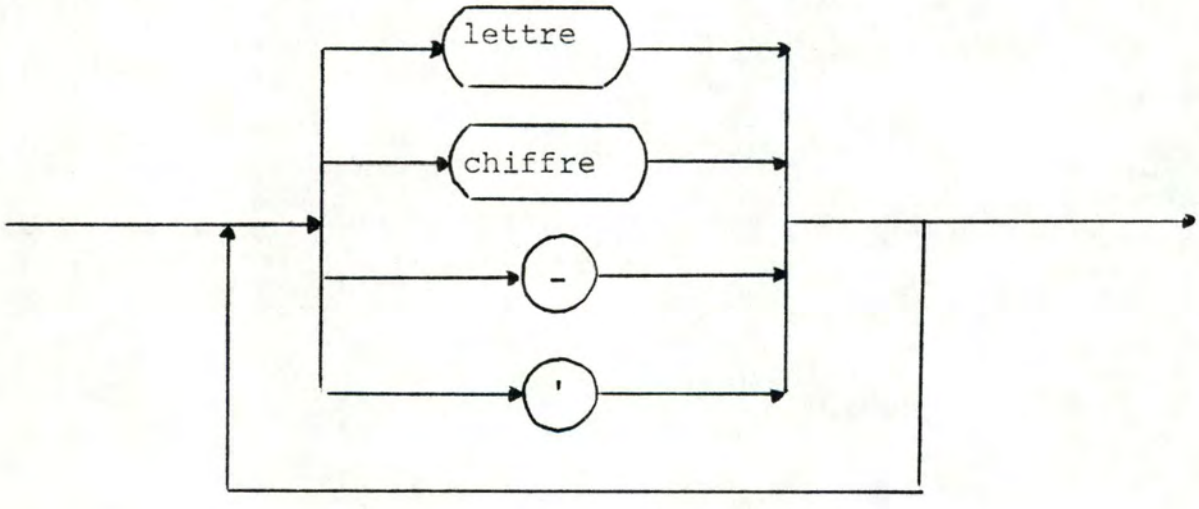
suite



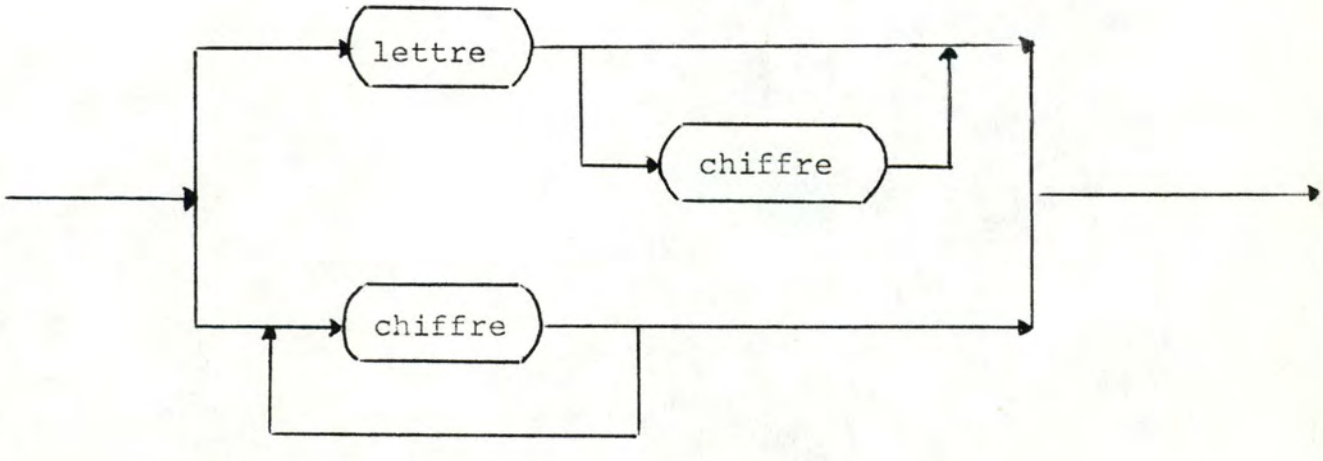
nom-d'ensemble/nom de sous-ensemble choisi



élément



nombre



Annexe 2. Messages d'erreur de l'analyseur du langage.

1 : espace attendu.
2 : "=" ou "#" attendus.
3 : fin de fichier inattendue.
4 : "(" attendu.
5 : "%" attendu suivi d'un nom d'ensemble.
6 : fin de ligne attendue.
7 : lettre, chiffre, "-" ou "'" attendus.
8 : "=" attendu.
9 : ".." attendu.
10 : "+" attendu.
11 : ")" attendu.
12 : ")" ou "," attendus.
13 : tout ou nombre attendus.
14 : "&", "'", ":", """, "!", "*" ou ";" attendus.
15 : ";" attendu.
16 : "]" attendu.
17 : "]" ou "," attendus.
18 : ")" ou " " attendus.
19 : "+" ou "-" attendus.
20 : "[" ou "(" attendus.
21 : ":" attendu.
22 : avec attendu.
23 : de attendu.
24 : et attendu.
25 : placer ou répartir attendus.
26 : dans, avec, "M" ou "R" attendus.
27 : ";", ".", "M" ou "R" attendus.
28 : tout interdit ici.

101 : 2 ensembles portent le même nom.
102 : 2 éléments dans un même ensemble portent le même nom.
103 : 2 suites variables portent le même nom.
104 : identificateur composé d'1 lettre ou d'1 chiffre attendu.

111 : cet ensemble n'a pas été déclaré.
112 : utilisation de sous-ensemble choisi non encore créé.
113 : cet élément n'appartient pas à l'ensemble principal.
114 : il n'existe pas d'élément d'ordre si élevé dans l'ensemble principal.

121 : Si on n'utilise ni les partitions, ni le type d'action "répartir", le nombre d'objets choisis doit être égal au nombre de cases choisies.
122 : Si on n'utilise pas les objets pouvant être répétés, le nombre d'éléments choisis doit être inférieur au nombre d'éléments susceptibles d'être choisis.
123 : Le calcul des éléments susceptibles d'être choisis donne un résultat < 1.
124 : La borne inférieure de l'intervalle doit être inférieure à la borne supérieure.
125 : Il y a plus d'objets précisés que d'objets choisis.
126 : Le nombre d'objets précisés est supérieur au nombre d'objets que comprend l'ensemble.
127 : Il faut additionner les cases contenant le même nombre d'objets.
128 : Le total du nombre d'objets de la partition est différent du total du nombre d'objets choisis.
129 : Le total du nombre de cases de la partition est différent du total du nombre de cases choisies.

131 : l'ensemble précisant l'ensemble secondaire principal n'y est pas inclu.
132 : l'ensemble d'objets est utilisé comme ensemble de cases ou vice-versa.
133 : le sous-ensemble choisi a été nommé pour des objets et est utilisé pour des cases ou vice-versa.
134 : l'ensemble ayant servi à créer le sous-ensemble choisi est différent de celui pour lequel on veut maintenant l'utiliser.
135 : les éléments appartenant à un ensemble d'objets identiques ne peuvent être cités.

136 : on ne peut définir de suite pour un ensemble où les éléments n'ont pas été cités.
137 : la précision d'objets doit être utilisée avec un ensemble principal secondaire.

201 : les objets pouvant être répétés peuvent uniquement se placer dans des cases indistingables ou totalement distingables.
202 : les objets identiques peuvent uniquement se placer dans des cases totalement distingables.

211 : un ensemble primaire composant un ensemble secondaire doit comprendre des objets ne pouvant être répétés.
212 : un ensemble primaire composant un ensemble secondaire doit être déclaré sans citer ses éléments.
213 : on ne peut créer une suite ordonnée inclue dans un ensemble secondaire.
214 : on ne peut créer une suite inclue dans un ensemble secondaire.

221 : les partitions s'utilisent uniquement avec des cases totalement distingables ou indistingables.
222 : les partitions s'utilisent uniquement avec des objets différents et pouvant être répétés.
223 : les partitions s'utilisent uniquement avec le type d'action "placer".
224 : les partitions ne s'utilisent pas dans une action où est utilisée la précision d'objets

231 : la précision d'objets s'utilise uniquement avec le type d'action "placer".
232 : l'ensemble secondaire principal de la précision d'objets ne peut comprendre des objets identiques.
233 : les ensembles précisant un ensemble secondaire doivent comprendre des objets différents et ne pouvant être répétés.
234 : l'utilisation de la précision d'ensembles exige des cases indistingables.

235 : le type d'action "répartir" exige des cases totalement distingables.
236 : le type d'action "répartir" exige des objets différents et ne pouvant être répétés.
237 : il n'est pas possible de mélanger les "au moins" et "au plus" dans la précision d'objets.

301 : il ne peut y avoir plus de 10 actions.
302 : il ne peut y avoir plus de 15 ensembles primaires.
303 : il ne peut y avoir plus de 10 ensembles secondaires.
304 : il ne peut y avoir plus de 15 sous-ensembles choisis.
305 : il ne peut y avoir plus de 10 ensembles composant un ensemble secondaire.
306 : il ne peut y avoir plus de 10 couples pour une partition.
307 : il ne peut y avoir plus de 5 sous-ensembles choisis dans une suite variable.
308 : il ne peut y avoir plus de 5 ensembles précisant un ensemble secondaire.
309 : il ne peut y avoir plus de 10 éléments cités dans un ensemble primaire.
310 : il ne peut y avoir plus de 10 éléments cités dans une suite ordonnée.
311 : il ne peut y avoir plus de 10 éléments cités dans une suite.
312 : cet identificateur comprend plus de 30 caractères.

401 : la possibilité d'utiliser des nombres variables n'est pas encore implémentée.

Annexe 3. Exercices traduits dans le langage.

Exercice n° 20 [3 ex. 2.4 p.24]

En supposant qu'il n'y a pas de répétitions, (i) combien de nombres de 3 chiffres peut-on former à l'aide des six chiffres 2, 3, 5, 6, 7 et 9 ? (ii) Combien de ces nombres sont inférieurs à 400 ? (iii) Combien sont pairs ? (iv) Combien sont impairs ? (v) Combien sont des multiples de 5 ?

Exercice n° 21 [3 ex. 2.6 p.25]

(i) De combien de façons différentes, 3 garçons et 2 filles peuvent-ils prendre place sur un banc ? (ii) De combien de façons peuvent-ils s'asseoir si les garçons s'assoient les uns à côté des autres et s'il en est de même pour les filles ? (iii) De combien de manières différentes peuvent-ils s'asseoir si seulement les filles s'assoient l'une à côté de l'autre ?

Exercice n° 22 [3 ex. 2.7 p.25]

Chaque signal consistant de 6 pavillons alignés, combien de signaux différents peut-on former à l'aide de 4 pavillons rouges et 2 pavillons bleus ?

Exercice n° 23 [3 ex. 2.8 p.25]

Combien de permutations distinctes peut-on former avec toutes les lettres des mots : (i) leur, (ii) anabase, (iii) sociologique ?

Exercice n° 24 [3 ex. 2.20 p.28]

De combien de manières peut-on former un jury de 3 hommes et 2 femmes parmi 7 hommes et 5 femmes ?

Exercice n° 25 [3 ex. 2.22 p.29]

A l'oral d'un examen, un étudiant doit répondre à 8 questions sur un total de 10. (i) Combien de choix possibles y a-t-il ? (ii) Combien de choix y a-t-il s'il doit répondre aux 3 premières questions ? (iii) Combien de choix y a-t-il s'il doit

répondre au moins à 4 des 5 premières questions ?

Exercice n° 26 [3 ex. 2.24 p.29]

De combien de manières différentes, un professeur peut-il choisir un ou plusieurs étudiants parmi six étudiants éligibles ?

Exercice n° 27 [3 ex. 2.35 p.31]

Il y a 6 chemins possibles allant de A à B et 4 chemins allant de B à C. (i) De combien de façons peut-on aller de A à C en passant par B ? (ii) De combien de façons peut-on aller et revenir de A à C en passant par B ? (iii) De combien de façons peut-on aller et revenir de A à C en ne passant qu'une seule fois par le même chemin ?

EXERCICE 20

```
%6-CHIFFRES = (2,3,5,6,7,9);
%CHIFFRES-DU-NOMBRE # 3;
PLACER 3 DE %6-CHIFFRES DANS TOUT DE %CHIFFRES-DU-NOMBRE.
%6-CHIFFRES = (2,3,5,6,7,9);
%CHIFFRES-DU-NOMBRE # 3;
PLACER %I = 1 DE %6-CHIFFRES R (2,3) DANS TOUT DE %CHIFFRES-DU-NOMBRE R [1];
PLACER 2 DE %6-CHIFFRES M (%I) DANS TOUT DE %CHIFFRES-DU-NOMBRE R [2,3].
%6-CHIFFRES = (2,3,5,6,7,9);
%CHIFFRES-DU-NOMBRE # 3;
PLACER %I = 1 DE %6-CHIFFRES R (2,6) DANS TOUT DE %CHIFFRES-DU-NOMBRE R [3];
PLACER 2 DE %6-CHIFFRES M (%I) DANS TOUT DE %CHIFFRES-DU-NOMBRE R [1,2].
%6-CHIFFRES = (2,3,5,6,7,9);
%CHIFFRES-DU-NOMBRE # 3;
PLACER %I = 1 DE %6-CHIFFRES M (2,6) DANS TOUT DE %CHIFFRES-DU-NOMBRE R [3];
PLACER 2 DE %6-CHIFFRES M (%I) DANS TOUT DE %CHIFFRES-DU-NOMBRE R [1,2].
%6-CHIFFRES = (2,3,5,6,7,9);
%CHIFFRES-DU-NOMBRE # 3;
PLACER TOUT DE %6-CHIFFRES R (5) DANS TOUT DE %CHIFFRES-DU-NOMBRE R [3];
PLACER 2 DE %6-CHIFFRES M (5) DANS TOUT DE %CHIFFRES-DU-NOMBRE R [1,2].
```

EXERCICE 21

```
%GARCONS # 3;
%FILLES # 2;
%PLACES-SUR-LE-BANC # 5;
%ENFANTS = %GARCONS + %FILLES;
PLACER TOUT DE %ENFANTS DANS TOUT DE %PLACES-SUR-LE-BANC.
```

```
GARCONS # 3;
FILLES # 2;
PLACES-SUR-LE-BANC # 5;
LACER TOUT DE %GARCONS DANS TOUT DE %PLACES-SUR-LE-BANC R [1..3];
LACER TOUT DE %FILLES DANS TOUT DE %PLACES-SUR-LE-BANC R [4,5];
U PLACER TOUT DE %GARCONS DANS TOUT DE %PLACES-SUR-LE-BANC R [3..5];
LACER TOUT DE %FILLES DANS TOUT DE %PLACES-SUR-LE-BANC R [1,2].
GARCONS # 3;
FILLES # 2;
PLACES-SUR-LE-BANC # 5;
LACER TOUT DE %FILLES DANS TOUT DE %PLACES-SUR-LE-BANC R [1,2];
LACER TOUT DE %GARCONS DANS TOUT DE %PLACES-SUR-LE-BANC R [3..5];
U PLACER TOUT DE %FILLES DANS TOUT DE %PLACES-SUR-LE-BANC R [2,3];
LACER TOUT DE %GARCONS DANS TOUT DE %PLACES-SUR-LE-BANC R [1,4,5];
U PLACER TOUT DE %FILLES DANS TOUT DE %PLACES-SUR-LE-BANC R [3,4];
LACER TOUT DE %GARCONS DANS TOUT DE %PLACES-SUR-LE-BANC R [1,2,5];
U PLACER TOUT DE %FILLES DANS TOUT DE %PLACES-SUR-LE-BANC R [4,5];
LACER TOUT DE %GARCONS DANS TOUT DE %PLACES-SUR-LE-BANC R [1..3].


EXERCICE 22
-----------

%PAVILLONS-ROUGES # 4";
%PAVILLONS-BLEUS # 2";
%PLACE-D'UN-PAVILLON # 6;
%PAVILLONS = %PAVILLONS-ROUGES + %PAVILLONS-BLEUS;
PLACER TOUT DE %PAVILLONS DANS TOUT DE %PLACES-D'UN-PAVILLON.


EXERCICE 23
-----------

%LETTRES # 4;
%LETTRES-DU-MOT # 4;
PLACER TOUT DE %LETTRES DANS TOUT DE %LETTRES-DU-MOT.
```

```
%LETTRES-DIFFERENTES # 4;
%LETTRES-A # 3";
%LETTRES-DU-MOT # 4;
%LETTRES = %LETTRES-DIFFERENTES + %LETTRES-A;
PLACER TOUT DE %LETTRES DANS TOUT DE %LETTRES-DU-MOT.
%LETTRES-DIFFERENTES # 7;
%LETTRES-O # 3";
%LETTRES-I # 2";
%LETTRES-DU-MOT # 12;
%LETTRES = %LETTRES-DIFFERENTES + %LETTRES-O + %LETTRES-I;
PLACER TOUT DE %LETTRES DANS TOUT DE %LETTRES-DU-MOT.
```

EXERCICE 24

```
%HOMMES # 7;
%FEMMES # 5;
%JURES # 5!;
%PERSONNES = %HOMMES + %FEMMES;
PLACER 5 DE %PERSONNES AVEC 3 DE %HOMMES DANS TOUT DE %JURES.
```

EXERCICE 25

```
%QUESTIONS # 10;
%QUESTIONS-POSEES # 8!;
PLACER 8 DE %QUESTIONS DANS TOUT DE %QUESTIONS-POSEES.
%QUESTIONS # 10;
%QUESTIONS-POSEES # 8!;
PLACER 3 DE %QUESTIONS R [1..3] DANS %I = 3 DE %QUESTIONS-POSEES;
PLACER 5 DE %QUESTIONS M [1..3] DANS TOUT DE %QUESTIONS-POSEES M (%I).
%QUESTIONS # 10;
%QUESTIONS-POSEES # 8!;
PLACER 4 DE %QUESTIONS R [1..5] DANS %I = 4 DE %QUESTIONS-POSEES;
PLACER 4 DE %QUESTIONS R [6..10] DANS TOUT DE %QUESTIONS-POSEES M (%I);
OU PLACER TOUT DE %QUESTIONS R [1..5] DANS %I = 5 DE %QUESTIONS-POSEES;
PLACER 3 DE %QUESTIONS R [6..10] DANS TOUT DE %QUESTIONS-POSEES M (%I).
```

XERCICE 26

ETUDIANTS # 6;

ETUDIANTS-ELUS # 6!;

LACER 1 DE %ETUDIANTS DANS 1 DE %ETUDIANTS-ELUS;

U PLACER 2 DE %ETUDIANTS DANS 2 DE %ETUDIANTS-ELUS;

U PLACER 3 DE %ETUDIANTS DANS 3 DE %ETUDIANTS-ELUS;

U PLACER 4 DE %ETUDIANTS DANS 4 DE %ETUDIANTS-ELUS;

U PLACER 5 DE %ETUDIANTS DANS 5 DE %ETUDIANTS-ELUS;

U PLACER 6 DE %ETUDIANTS DANS 6 DE %ETUDIANTS-ELUS.

XERCICE 27

CHEMINS-DE-A-A-B # 6;

CHEMINS-DE-B-A-C # 4;

PARTIES-DU-CHEMIN # 2;

LACER 1 DE %CHEMINS-A-A-B DANS TOUT DE %PARTIES-DU-CHEMIN R [1];

LACER 1 DE %CHEMINS-B-A-C DANS TOUT DE %PARTIES-DU-CHEMIN R [2].

CHEMINS-DE-A-A-B # 6;

CHEMINS-DE-B-A-C # 4;

PARTIES-DU-CHEMIN # 4;

LACER 1 DE %CHEMINS-A-A-B DANS TOUT DE %PARTIES-DU-CHEMIN R [1];

LACER 1 DE %CHEMINS-B-A-C DANS TOUT DE %PARTIES-DU-CHEMIN R [2];

LACER 1 DE %CHEMINS-B-A-C DANS TOUT DE %PARTIES-DU-CHEMIN R [3];

LACER 1 DE %CHEMINS-A-A-B DANS TOUT DE %PARTIES-DU-CHEMIN R [4].

CHEMINS-DE-A-A-B # 6;

CHEMINS-DE-B-A-C # 4;

PARTIES-DU-CHEMIN # 4;

LACER %I = 1 DE %CHEMINS-A-A-B DANS TOUT DE %PARTIES-DU-CHEMIN R [1];

LACER %J = 1 DE %CHEMINS-B-A-C DANS TOUT DE %PARTIES-DU-CHEMIN R [2];

LACER 1 DE %CHEMINS-B-A-C M (%J) DANS TOUT DE %PARTIES-DU-CHEMIN R [3];

LACER 1 DE %CHEMINS-A-A-B M (%I) DANS TOUT DE %PARTIES-DU-CHEMIN R [4].

Annexe 4. Définitions du modèle entité-association de l'exercice.

ENTITE objet

IDENTIFIE PAR numéro-objet et numéro-ens-O. (objet/ens.objets: objet)

PROPRIETES nom-objet
numéro-objet

ENTITE ensemble-objets

SYNONYME set-O.

IDENTIFIE PAR numéro-ens-O.
nom-ens-O.

PROPRIETES nom-ens-O.
numéro-ens-O.
type-ens-O.
définition-card-O.

ENTITE action

IDENTIFIE PAR numéro-action

PROPRIETES numéro-action
type-action
coordination

ENTITE sous-ensemble-choisi-O.

SYNONYME subsut

IDENTIFIE PAR numéro-sech-O.
nom-sech-O.

PROPRIETES numéro-sech-O.
nom-sech-O.

Note : case a une définition analogue à objet,
ensemble-cases a une définition analogue à ensemble-objets,
sous-ensemble-choisi-C. a une définition analogue à sous-ensemble-choisi-O.

ASSOCIATION objet/ens-objets

ASSOCIE objet
ens.objets

CONNECTIVITE 1-1 pour objet
1-* pour ens-objets

ASSOCIATION est-inclu/contient-O.

DESCRIPTION exprime qu'un ensemble est inclu dans un ensemble secondaire

ASSOCIE ensemble-objets
ensemble-objets

CONNECTIVITE 0-10

ASSOCIATION expression-objets

PROPRIETES nombre-objets-ch
nombre-objets-pos
restriction
dés-se-O.-M
dés-se-O.-R

ASSOCIE ensemble-objets
action

CONNECTIVITE 0-10 pour ensemble-objets
1-1 pour action

ASSOCIATION action/sech-O.

DESCRIPTION exprime la création par une action d'un sous-ensemble choisi d'objets.

ASSOCIE action
sous-ensemble-choisi-O.

CONNECTIVITE 0-1 pour action
1-1 pour sous-ensemble-choisi-O.

Note : case/ens-cases a une définition analogue à objet/ens-objets
est-inclu/contient-C. a une définition analogue à
est-inclu/contient-O.,
expresssion-cases a une définition analogue à expression-objets,
action/sech-C. a une définition analogue à action/sech-O..

ASSOCIATION précision

DESCRIPTION exprime qu'un ensemble d'objets précise un ensemble principal dans une action

PROPRIETES nbopr
signopr

ASSOCIE ensemble-objets
action

CONNECTIVITE 0-10 pour ensemble-objets
0-5 pour action

ASSOCIATION partition

PROPRIETES couplepart *

ASSOCIE ensemble-cases
action

CONNECTIVITE 0-10 pour ensemble-cases
0-1 pour action

PROPRIETE restriction-O.

SYNONYME restr

FORMAT (absenteO., appO., non appO., appetnonappO.)

DESCRIPTION restriction-O. = absenteO. si l'ensemble principal n'est pas restreint
restriction-O. = appO. si l'ensemble principal est restreint par
R suivi d'un sous-ensemble créé.
restriction-O. = nonappO. si l'ensemble principal est restreint par
M suivi d'un sous-ensemble créé.
restriction-O. = appetnonappO. si l'ensemble principal est restreint à la fois par R et M.

Note : restriction-C. a une définition analogue.

GROUPE dés-se-O.-M

SYNONYME sousens-O.-M

DESCRIPTION représente un sous-ensemble d'objets suivant le signe "M"
est inexistant si restr-O. = absente ou appO.

CONTIENT type-sousens-O.-M
eltsuite-O.-M *
eltsuiteord-O.-M *
déssuitevar-O.-M

Note : dés-se-O.-R, dés-se-C.-M, dés-se-C.-R ont des définitions analogues.

PROPRIETE type-sousens-O.-M

FORMAT (suiteom, suiteordom, suitevarom)

DESCRIPTION type-sousens-O.-M
= suiteom si le sous-ensemble créé est une suite,
= suiteordom si le sous-ensemble créé est une suite ordonnée,
= suitevarom si le sous-ensemble créé est une suite variable.

PROPRIETE eltsuite-O.-M

DESCRIPTION eltsuite-O.-M représente l'ordre dans lequel un élément de la suite a été déclaré;

si type-sousens-O.-M ≠ suiteom alors eltsuite-O.-M est inexistant;

si la suite est créée en donnant les bornes inférieures et supérieures, il y aura 3 eltsuite-O.-M; le premier sera la borne inférieure, le second prendra arbitrairement la valeur zéro, le troisième sera la borne supérieure.

PROPRIETE eltsuiteord-O.-M

FORMAT entier

DESCRIPTION eltsuiteord-O.-M est l'élément de la suite ordonnée.

si type-sousens-O.-M ≠ suiteordom alors eltsuiteord-O.-M. est inexistant

si la suite ordonnée est créée en donnant les bornes inférieures et supérieures, il y aura 3 eltsuiteord-O.-M : le premier sera la borne inférieure, le second zéro, le troisième la borne supérieure.

GROUPE déssuitevar-O.-M

DESCRIPTION représente une suite variable.
si type-sousens-O.-M ≠ suitevarom, alors déssuitevar-O.-M est inexistant

CONTIENT defdep-O.-M
désignation-sech-dep-O.-M
eltsuitevar-O.-M *

PROPRIETE defdep-O.-M

FORMAT booléen

DESCRIPTION defdep = vrai si la suite variable a la forme b. (cf. p.)
defdep = faux si la suite variable a la forme a. (cf. p.)

GROUPE désignation-sech-dep-O.-M

DESCRIPTION représente une suite variable de la forme b., est inexistant si defdep = faux.

CONTIENT ssbase-O.-M
plus-O.-M
dep-O.-M

PROPRIETE eltsuitevar-O.-M

FORMAT entier

DESCRIPTION donne numéro-sech-O. du sous-ensemble-choisi
si defdep = vrai, eltsuitevar-O.-M est inexistant

PROPRIETE ssbase-O.-M

FORMAT entier

DESCRIPTION donne numéro-sech-O.- du sous-ensemble-choisi

PROPRIETE plus-O.-M

FORMAT booléen

DESCRIPTION plus-O.-M = vrai si le sens dans lequel le sous-ensemble-choisi doit être déplacé pour obtenir la suite variable est la droite;
plus-O.-M = faux sinon

PROPRIETE dep-O.-M

FORMAT entier

DESCRIPTION représente le nombre d'éléments dont il faut déplacer le sous-ensemble choisi pour obtenir la suite variable.

PROPRIETE nombre-objets-ch

SYNONYME nboch

FORMAT entier

PROPRIETE nombre-objets-pos

SYNONYME nbop

FORMAT entier

DESCRIPTION désigne le nombre d'objets susceptibles d'être choisis

Note : nombre-cases-ch et nombre-cases-pos ont des définitions analogues.

GROUPE couplepart

DESCRIPTION représente un couple d'une partition.

CONTIENT nombre-objets-dans-une-boîte
nombre-fois-même-nombre-objets

PROPRIETE nombre-objets-dans-une-boîte

SYNONYME nbodsb

FORMAT entier

PROPRIETE nombre-fois-même-nombre-objets

SYNONYME nbfmembo

FORMAT entier

DESCRIPTION représente le nombre de cases contenant le même nombre d'objets

PROPRIETE nbopr

FORMAT entier

DESCRIPTION représente le nombre d'objets à choisir dans l'ensemble composant l'ensemble principal

PROPRIETE signopr

FORMAT (plus, moins, existpas)

DESCRIPTION signale si nbopr représente le nombre exact (signopr = existpas), le minimum (signopr = moins) ou le maximum (signopr = plus) d'objets à choisir dans l'ensemble composant l'ensemble principal.

PROPRIETE numéro-action

FORMAT entier

PROPRIETE type-action

SYNONYME typact

FORMAT booléen

DESCRIPTION type-action = vrai si l'action est du type "placer"
type-action = faux si l'action est du type "répartir"

PROPRIETE coordination

SYNONYME coord

FORMAT (et, ou, sans)

DESCRIPTION coordination = ou si le mot réservé "placer" est précédé de "ou",
= sans si le mot réservé "placer" est précédé de "sans",
= et sinon.

PROPRIETE nom-ens-O.

FORMAT X(30)

PROPRIETE numéro-ens-O.

FORMAT entier

PROPRIETE type-ens-O.

FORMAT (ossrep, orep, oident)

DESCRIPTION type-ens-O. = ossrep si l'ensemble d'objets est du type 4 (p. 34),
= orep si l'ensemble d'objets est du type 2 (p. 34),
= oident si l'ensemble d'objets est du type 3 (p. 34),
type-ens-O. est inexistant si l'ensemble est un ensemble secondaire.

PROPRIETE définition-card-O.

SYNONYME defcomp-O.

FORMAT booléen

DESCRIPTION définition-card-O. = vrai si l'on ne cite pas les éléments de l'ensemble,
= faux sinon.
définition-card-O. est inexistant si l'ensemble est un ensemble secondaire.

PROPRIETE nom-sech-O.

SYNONYME nomsubsut-O.

FORMAT X(30)

PROPRIETE numéro-sech-O.
FORMAT entier

PROPRIETE numéro-objet
FORMAT entier

PROPRIETE nom-objet
FORMAT X(30)
DESCRIPTION nom-objet est inexistant si définition-card-O.
de l'ensemble auquel il appartient = vrai;
si l'on cite uniquement une borne inférieure et une borne supérieure,
nom-objet de numéro-objet =
1. est la borne inférieure
2. prend arbitrairement la valeur '.'
3. est la borne supérieure.

PROPRIETE type-ens-C.
FORMAT (bssord, bord, bcire, bsym, bsymcire)
DESCRIPTION type-ens-C. est inexistant si l'ensemble est un ensemble secondaire
type-ens-C. = bssord si l'ensemble de cases est du type 2 p. 54
= bord si l'ensemble de cases est du type 1 p. 54
= bcire si l'ensemble de cases est du type 3 p. 54
= bsym si l'ensemble de cases est du type 4 p. 54
= bsymcire si l'ensemble de cases est du type 5 p. 54

Note : nom-ens-c. a une définition analogue à nom-ens-O.
numéro-ens-C. a une définition analogue à numéro-ens-O.
définition-card-C. a une définition analogue à définition-card-O.
nom-sech-C. a une définition analogue à nom-sech-O.
numéro-sech-C. a une définition analogue à numéro-sech-O.

Annexe 5. Spécifications des modules.

analyse

ARGUMENTS texte d'un exercice en langage formalisé. (profelf)

PRECONDITIONS la tête de lecture est sur le premier caractère du texte (c'est-à-dire que le prochain caractère lu sera le premier caractère du texte)

RESULTATS si le texte est correct, alors intsol et intestrd
sinon, proferr

traddecl

ARGUMENTS texte d'un exercice en langage formalisé (profelf)

PRECONDITIONS la tête de lecture est sur le premier caractère du texte
numéro-de-ligne = 1 (numéro de la ligne qu'on est en train d'analyser)
numéro-de-caractère = 0 (numéro du caractère qu'on est en train d'analyser)

RESULTATS si les déclarations sont correctes, traduction de celles-ci.
sinon, proferr

POSTCONDITIONS la tête de lecture est sur le second caractère de la première ligne d'exercice ne commençant pas par "%"
type-ens-C. et type-ens-O. n'ont pas de valeur si ";" suit directement ")" ou un nombre.
numéro-de-ligne = nombre d'ensembles + 1
numéro-de-caractère = 1
nombre-d'ensembles-p = nombre d'ensembles primaires
nombre-d'ensembles-s = nombre d'ensembles secondaires

tradact

ARGUMENTS texte d'un exercice en langage formalisé
traduction des déclarations

PRECONDITIONS la tête de lecture est sur le second caractère de la première ligne d'exercice ne commençant pas par "%"
numéro-de-ligne = nombre d'ensembles + 1
numéro-de-caractère = 1
nombre-d'ensembles-p = nombre d'ensembles primaires
nombre-d'ensembles-s = nombre d'ensembles secondaires

RESULTATS si le texte est correct, alors intsol et intestrd
sinon, proferr

traddp

ARGUMENTS texte d'un exercice en langage formalisé (profelf)

PRECONDITIONS la tête de lecture est sur le premier caractère du texte.
numéro-de-ligne = 1
numéro-de-caractère = 0
nombre-d'ensembles-p = 0
nombre-d'ensembles-s = 0
nombre-de-sous-ensemble-ch = 0
fin-déclarations = faux

RESULTATS si les déclarations primaires sont correctes,
traduction de celles-ci.
sinon, proferr

POSTCONDITIONS type-ens-C. et type-ens-O. n'ont pas de valeur si ";" suit directement ")" ou un nombre
numéro-de-ligne = nombre d'ensembles primaires + 1
nombre-d'ensembles-p = nombre d'ensembles primaires
nombre-de-sous-ensemble-ch = 0
si fin-déclarations = vrai, alors
. la tête de lecture est sur le second caractère de la première ligne d'exercice ne commençant pas par "%"
. numéro-de-caractère = 1
. nombre-d'ensembles-1 = 0
sinon
. la tête de lecture est sur le second caractère du premier nom d'ensemble composant le premier ensemble secondaire
. numéro-de-caractère = rang dans la ligne du caractère précédant la tête de lecture
nombre-d'ensembles-s = 0

tradds

ARGUMENTS texte...

PRECONDITIONS . la tête de lecture est sur le second caractère du premier nom d'ensemble composant le premier ensemble secondaire
. numéro-de-caractère = rang dans la ligne du caractère précédant la tête de lecture
. nombre-d'ensembles-s = 0
nombre-d'ensembles-p = nombre d'ensembles primaires
fin-déclarations = faux
numéro-de-ligne = nombre d'ensembles primaires + 1

RESULTATS si les déclarations sont correctes, traduction de celles-ci
sinon, proferr

POSTCONDITIONS la tête de lecture est sur le second caractère de la première ligne d'exercice ne commençant pas par "%"
numéro-de-ligne = nombre d'ensembles + 1
numéro-de-caractère = 1

tradldp

ARGUMENTS texte ...

PRECONDITIONS . la tête de lecture est sur le second caractère suivant l'espace placé après "#" ou "="
. numéro-de-caractère = rang (tête de lecture) - 1
numéro-de-ligne = nombre d'ensembles primaires déjà rencontrés + 1
. déclarations-secondaires = faux
. fin-déclarations = faux
. nombre-d'ensembles-p = nombre d'ensembles primaires déjà rencontrés
. ddefcomp = vrai si on a rencontré "#"
= faux sinon

RESULTATS si la (numéro-de-ligne) ème déclaration d'un ensemble primaire est correcte, traduction des (numéro-de-ligne) premières déclarations; sinon, proferr

POSTCONDITIONS . numéro-de-ligne = numéro-de-ligne + 1
. nombre-d'ensembles-p = nombre-d'ensembles-P + 1
. si fin-déclarations = vrai alors la tête de lecture est sur le second caractère de la première ligne d'exercice ne commençant pas par "%" et numéro-de-caractère = 1
sinon, si déclarations-secondaires = vrai alors la tête de lecture est sur le second caractère du premier nom d'ensemble composant le premier ensemble secondaire et numéro-de-caractère = rang (tête de lecture) - 1
sinon, . la tête de lecture est sur le second caractère suivant l'espace placé après "#" ou "=" et numéro-de-caractère = rang (tête de lecture) -1
. ddefcomp = vrai si on a rencontré "#"
= faux sinon

tradlds

ARGUMENTS texte ...

PRECONDITIONS . la tête de lecture est sur le second caractère du premier nom d'ensemble composant un ensemble secondaire
. numéro-de-caractère = rang (tête de lecture) - 1

. numéro-de-ligne = nombre-d'ensembles-p + nombre d'ensembles primaires déjà rencontrés + 1
. fin-déclarations = faux
. nombre-d'ensembles-s = nombre d'ensembles secondaires déjà rencontrés

RESULTATS si la (numéro-de-ligne) ème déclaration d'un ensemble secondaire est correcte, traduction des (numéro-de-ligne) premières déclarations;
sinon proferr.

POSTCONDITIONS . numéro-de-ligne = numéro-de-ligne + 1
. nombre-d'ensembles-s = nombre d'ensembles-s + 1
si fin-déclarations = vrai alors la tête le second caractère de la première ligne d'exercice ne commençant pas par "%" et numéro-de-caractère = 1
sinon, la tête de lecture est sur le second caractère du premier nom d'ensemble composant le premier ensemble secondaire et numéro-de-caractère = rang (tête de lecture)-1

tradlact

ARGUMENTS texte ...

PRECONDITIONS . la tête de lecture est sur le second caractère d'une ligne d'action
. numéro-de-caractère = 1
. numéro-de-ligne = nombre d'ensembles + nombre d'actions déjà rencontrées
. fin-texte = faux
. nombre-d'actions = nombre d'actions déjà rencontrées

RESULTATS si l'action en cours est correcte, traduction des déclarations et des nombre-d'actions premières déclarations; sinon proferr

POSTCONDITIONS . nombre-d'actions = nombre-d'actions + 1
. si fin-texte = vrai alors
la tête de lecture est sur le caractère suivant le "."
numéro-de-ligne = numéro-de-ligne
numéro-de-caractère = rang (".")
sinon la tête de lecture est sur le second caractère d'une ligne d'action
numéro-de-caractère = 1
numéro-de-ligne = numéro-de-ligne + 1
fin-texte = faux

(*$S+*)

Annexe 6. Programmes des modules.

```
PROGRAM EADANCOMB;


CONST NBMAXACTION    =  10;
      NBMAXSP        =  15;
      NBMAXSS        =  10;
      NBMAXSUBSUT    =  15;
      NBMAXSCSS      =  10;
      NBMAXPARDES    =  10;
      NBMAXSSLIMACT  =   5;
      NBMAXSPSOBJ    =   5;
      NBMAXCDES      =  10;
      NBMAXCSORD     =  10;
      NBMAXCSUITE    =  10;
      LMAXMOT        =  30;
      MOTBLANC       =  '                              ';
      (*MOTBLANC DOIT ABSOLUMENT CONTENIR LMAXMOT FOIS LE CARACTERE BLANC*)
      NBMAXCARSOLAC = 100;
      LMAXNOMBALPH   =   2;
      ALPHVIDE       ='  ';



TYPE SUBSET = RECORD

        CASE T : INTEGER OF

            1 : (NBCSUITE : INTEGER;
                 ELTSUITE : ARRAY[1..NBMAXCSUITE] OF INTEGER);
            2 : (NBCSORD : INTEGER;
                 ELTSORD : ARRAY[1..NBMAXCSORD] OF INTEGER);
            3 : (CASE DEFDEP : BOOLEAN OF
                TRUE  : (SSBASE : INTEGER; PLUS : BOOLEAN; DEP : INTEGER);
                FALSE : (ENSSS : ARRAY[1..NBMAXSSLIMACT] OF INTEGER;
                         NBSSLIMACT : 0..NBMAXSSLIMACT))
                END;


    RSGTSET = RECORD

        SSET : INTEGER;
        NBCCH,NBCP : INTEGER;
        NUMSSD : 0..NBMAXSUBSUT;
        RESTR : (ABSENTE, APP, NONAPP, APPETNONAPP);
        SOUSENS : ARRAY[FALSE..TRUE] OF SUBSET

              END;


    COUPLEPART = RECORD

        NBODSB, NBFMEMBO : INTEGER

              END;


    PARTITION = RECORD

        NBPART : 0..NBMAXPARDES;
        TPART : ARRAY[1..NBMAXPARDES] OF COUPLEPART

              END;
```

```
DDESSETPREC = RECORD

    SSSET    : INTEGER;
    NBOPR    : INTEGER;
    SIGNOPR  : (PLUS,MOINS,EXISTPAS)

               END;


TDESSETPREC = RECORD

    DSETPREC : ARRAY[1..NBMAXSPSOBJ] OF DDESSETPREC;
    NBSPSOBJ : 0..NBMAXSPSOBJ

               END;


TYPENOMBRE = (NUM,ALPH,TOUT);

NOMLVAR = STRING[LMAXMOT];

TRECH = (NC,INT,COMP);


TYPECAR = (CHIFFRE,LETTRE,TIRET,APOS,AUTRE);


TACTION = RECORD

    SOLUTION : INTEGER;
    COORD : (ET,OU,SANS);
    TYPACT : BOOLEAN;
    ENS : ARRAY[FALSE..TRUE] OF RSGTSET;
    PART : BOOLEAN;
    DESPART : PARTITION;
    SETOBJPREC : BOOLEAN;
    DESSETPREC : TDESSETPREC

          END;


TSETP = RECORD

    NOMSP : NOMLVAR;
    NBCSP : INTEGER;
    TYPSP : (INCONNU,OSSREP,OREP,OIDENT,BORD,BSSORD,BCIRC,BSYM,
             BSYMCIRC);
    DEFCOMP : BOOLEAN;
    COMPOSANT : ARRAY[1..NBMAXCDES] OF NOMLVAR

        END;


TSETS = RECORD

    NOMSS : NOMLVAR;
    SETC  : ARRAY[1..NBMAXSCSS] OF INTEGER;
     (* 0 INDIQUE QUE L'ON N'A PAS ENCORE TROUVE *)
    NBSCSS : 0..NBMAXSCSS;
        (*REPRESENTE LE NOMBRE DE SETS PRIMAIRES DEJA RENCONTRES
          DANS L'ANALYSE D'UNE LIGNE DE DECLARATION DE SET SECONDAIRE*)
    COMPOIDENT : BOOLEAN

        END;
```

```
    TSUBSUT = RECORD

        NOMSUBSUT : NOMLVAR;
        COBJETS   : BOOLEAN

              END;


VAR ACTION : ARRAY[1..NBMAXACTION] OF TACTION;

    SETP   : ARRAY[1..NBMAXSP] OF TSETP;

    SETS   : ARRAY[1..NBMAXSS] OF TSETS;

    SUBSUT : ARRAY[1..NBMAXSUBSUT] OF TSUBSUT;


    NBACTION : 0..NBMAXACTION;

    NBSETP   : 0..NBMAXSP;

    NBSETS   : 0..NBMAXSS;

    NBSUBSUT : 0..NBMAXSUBSUT;

(*************************************************************************)

    F : TEXT;

    C:  CHAR;

    MOT : NOMLVAR;

    NOMBRE : INTEGER;

    NOMBALPH : STRING[LMAXNOMBALPH];

    TYPEN : TYPENOMBRE;

    NUMLIGNE,NUMCAR : INTEGER;

    I,J,K,L : INTEGER;  (*EXISTENT UNIQUEMENT POUR TESTER*)

    B : BOOLEAN;


(*************************************************************************)

PROCEDURE ERREUR(NUMERREUR:INTEGER);

BEGIN

    WRITELN('ERREUR ',NUMERREUR,' LIGNE ',NUMLIGNE,' CARACTERE ',NUMCAR);
    EXIT (PROGRAM)

END;

(*-------------------------------------------------------------------------*)
```

```
UNCTION TYPEC (CAR:CHAR) : TYPECAR;

EGIN

   IF (ORD(CAR) > 47) AND (ORD(CAR) < 58) THEN TYPEC:=CHIFFRE

                                          ELSE

      IF ((ORD(CAR) > 64) AND (ORD(CAR) < 91)) OR
         ((ORD(CAR) > 96) AND (ORD(CAR) < 123)) THEN TYPEC:=LETTRE

                                                 ELSE

         IF (ORD(C) = 45) THEN TYPEC:=TIRET

                         ELSE IF ORD(C) = 39 THEN TYPEC:=APOS

                                            ELSE TYPEC:=AUTRE

ND;
*-------------------------------------------------------------------------------*)

ROCEDURE LIREC;

EGIN

   IF EOF(F) THEN ERREUR(3)
             ELSE READ(F,C);
   NUMCAR:=NUMCAR+1

ND;
*-------------------------------------------------------------------------------*)

ROCEDURE LIREMOT;

AR I : 0..LMAXMOT;

EGIN

   MOT:=MOTBLANC;
   I:=0;
   WHILE TYPEC(C) <> AUTRE DO

   BEGIN

      IF I = LMAXMOT THEN ERREUR(312);
      I:=I+1;
      MOT[I]:=C;
      LIREC

   END;

   IF I = 0 THEN ERREUR(7);

   MOT:=COPY(MOT,1,I)

ND;
*-------------------------------------------------------------------------------*)
```

```
PROCEDURE LIRENOMBRE(VAR TYPENB : TYPENOMBRE);

BEGIN

   IF TYPEC(C) = CHIFFRE THEN

   BEGIN

      NOMBRE:=0;

      WHILE TYPEC(C) = CHIFFRE DO

      BEGIN

         NOMBRE:=NOMBRE*10+(ORD(C)-48);
         LIREC

      END;

      IF NOMBRE = 0 THEN ERREUR(13);
      TYPENB:=NUM

   END

                                          ELSE

   IF TYPEC(C) = LETTRE THEN

   BEGIN

      NOMBALPH:=ALPHVIDE;
      NOMBALPH[1]:=C;
      TYPENB:=ALPH;
      LIREC;
      NOMBALPH[2]:=C;
      IF C = ' ' THEN NOMBALPH:=COPY(NOMBALPH,1,1)
                 ELSE

      BEGIN

         IF NOMBALPH = 'TO' THEN

         BEGIN

            LIREC;
            IF C <> 'U' THEN ERREUR(13);
            LIREC;
            IF C <> 'T' THEN ERREUR(13);
            TYPENB:=TOUT

         END

                              ELSE

         IF TYPEC(C) <> CHIFFRE THEN ERREUR(13);

         LIREC;

      END

   END

                                                   ELSE ERREUR(13);

   IF TYPENB = ALPH THEN ERREUR(401)

END;
```

```
*----------------------------------------------------------------------------*)

UNCTION RANG(MOTRECHERCHE : NOMLVAR):INTEGER;

* CETTE FONCTION RECHERCHE LE RANG DU SET OU DU SUBSET NOMME MOTRECHERCHE.


  SI MOTRECHERCHE DESIGNE UN SET PRIMAIRE DEJA DECLARE,
  RANG = NUMERO DE LA LIGNE DE L'EXERCICE PRESENT DANS LEQUEL
         IL A ETE DECLARE

  SINON, SI MOTRECHERCHE DESIGNE UN NOM DE SET SECONDAIRE DEJA DECLARE,
         RANG = NUMERO DE LA LIGNE DE L'EXERCICE PRESENT DANS LEQUEL IL A
                ETE DECLARE - NOMBRES DE LIGNES CONCERNANT UNE DECLARATION
                DE SET PRIMAIRE + 100

         SINON, SI MOT RECHERCHE DESIGNE UN NOM DE SUBSET,
                RANG = ORDRE DANS LEQUEL LE SUBSET A ETE DECLARE + 200

                SINON, (MOTRECHERCHE NE CORRESPOND A AUCUN SET OU SUBSET)
                       RANG = 0
                                          *)

AR I,VIRANG : INTEGER;

EGIN

   VIRANG:=0;
   I:=1;

   WHILE (I <= NBSETP) AND (VIRANG = 0) DO

       IF MOTRECHERCHE = SETP[I].NOMSP THEN VIRANG:=I
                                       ELSE I:=I+1;

   IF VIRANG = 0 THEN

   BEGIN

       I:=1;

       WHILE (I <= NBSETS) AND (VIRANG = 0) DO

           IF MOTRECHERCHE = SETS[I].NOMSS THEN VIRANG:=I+100
                                           ELSE I:=I+1

   END;

   IF VIRANG = 0 THEN

   BEGIN

       I:=1;

       WHILE (I <= NBSUBSUT) AND (VIRANG = 0) DO

           IF MOTRECHERCHE = SUBSUT[I].NOMSUBSUT THEN VIRANG:=I+200

                                                 ELSE I:=I+1

   END;

   RANG:=VIRANG

ND;

*----------------------------------------------------------------------------*)
```

```
FUNCTION ECART (BI,BS : CHAR) : INTEGER;

VAR ORDALPHBS : INTEGER;

BEGIN

    IF TYPEC(BI) = TYPEC(BS) THEN ECART:=ORD(BS)-ORD(BI)+1

                             ELSE BEGIN

        IF ORD(BS) < 91 THEN ORDALPHBS:=ORD(BS)-64

                        ELSE ORDALPHBS:=ORD(BS)-96;

        ECART:=(ORDALPHBS+10)-(ORD(BI)-47)+1

    END

END;

{*-------------------------------------------------------------------------*}

FUNCTION RECHCOMP(TMOT:NOMLVAR;ISP:INTEGER):INTEGER;

VAR I : INTEGER;

    TROUVE : BOOLEAN;

BEGIN

    RECHCOMP:=0;
    TROUVE:=FALSE;
    I:=1;

    WHILE (I<=SETP[ISP].NBCSP) AND NOT TROUVE DO

    IF MOT = SETP[ISP].COMPOSANT[I]

        THEN BEGIN

                TROUVE:=TRUE;
                RECHCOMP:=I

             END

        ELSE I:=I+1

END;

(*-------------------------------------------------------------------------*)

FUNCTION RECHINT(TMOT:NOMLVAR;ISP:INTEGER):INTEGER;

BEGIN

    IF LENGTH(TMOT) <> 1 THEN ERREUR(104);
    IF (TYPEC(TMOT[1]) <> CHIFFRE) AND (TYPEC(TMOT[1]) <> LETTRE)
        THEN ERREUR(104);
    IF ORD(TMOT[1]) > ORD(SETP[ISP].COMPOSANT[3,1]) THEN ERREUR(113);
    RECHINT:=ECART(SETP[ISP].COMPOSANT[1,1],TMOT[1]);

END;

(*-------------------------------------------------------------------------*)
```

```
UNCTION RECHELT(TMOT:NOMLVAR;ISP:INTEGER;VAR TRENTREE:TRECH):INTEGER;

EGIN CASE TRENTREE OF

   COMP : RECHELT:=RECHCOMP(TMOT,ISP);

   INT  : RECHELT:=RECHINT(TMOT,ISP);

   NC   : IF SETP[ISP].COMPOSANT[2] = '.'

          THEN BEGIN

               RECHELT:=RECHINT(TMOT,ISP);
               TRENTREE:=INT

          END

          ELSE BEGIN

               RECHELT:=RECHCOMP(TMOT,ISP);
               TRENTREE:=COMP

          END

     END

ND;

*-------------------------------------------------------------------------------*)

**$I APPLE3:DECL*)

**$I APPLE3:ACTION1*)

**$I APPLE3:ACTION2*)

*-------------------------------------------------------------------------------*)



(*****************************************************************************
 *                                                                           *
 * DEBUT DU PROGRAMME PRINCIPAL                                              *
 *                                                                           *
 *****************************************************************************)

EGIN

   RESET(F,'APPLE3:ENONCE.TEXT');

   BEGIN

      NUMLIGNE:=1;NUMCAR:=0;

      TRADDECL;

      TRADACT

   END

ND.
```

```
PROCEDURE  TRADDECL;

VAR DECLSEC,FINDECL : BOOLEAN;

PROCEDURE TRADDP;

VAR DDEFCOMP : BOOLEAN;


PROCEDURE TRADLDP;

(*************************************************************************
*                                                                        *
* DEBUT DE LA TRADUCTION D'UNE LIGNE                                     *
*                                                                        *
*  DE DECLARATION PRIMAIRE                                               *
*                                                                        *
*************************************************************************)

BEGIN

    (*CONTROLE SI LE NOM DE SET N'EXISTERAIT PAS DEJA*)
    IF RANG(MOT) <> 0 THEN ERREUR(101);

    IF NBSETP = NBMAXSP THEN ERREUR(302);
    NBSETP:=NBSETP+1;

    WITH SETP[NBSETP] DO
    BEGIN

        DEFCOMP:=DDEFCOMP;
        NOMSP:=MOT;

        IF DEFCOMP THEN BEGIN

            LIRENOMBRE(TYPEN);
            IF TYPEN = NUM THEN NBCSP:=NOMBRE
                           ELSE ERREUR(28)

        END

        ELSE BEGIN

            IF C <> '(' THEN ERREUR(4);
            LIREC;
            LIREMOT;
            COMPOSANT[1]:=MOT;

            IF C = '.' THEN BEGIN

                LIREC;
                IF C <> '.' THEN ERREUR(9);
                COMPOSANT[2]:='.';
                IF LENGTH(MOT) <> 1 THEN ERREUR(104);
                IF (TYPEC(MOT[1]) <> LETTRE) AND (TYPEC(MOT[1]) <> CHIFFRE)
                    THEN ERREUR(104);
                LIREC;
                LIREMOT;
                IF LENGTH(MOT) <> 1 THEN ERREUR(104);
                IF (TYPEC(MOT[1]) <> LETTRE) AND (TYPEC(MOT[1]) <> CHIFFRE)
                    THEN ERREUR(104);
                COMPOSANT[3]:=MOT;
                IF ORD (COMPOSANT[1,1]) >= ORD (COMPOSANT[3,1])
                    THEN ERREUR(124);
                NBCSP:=ECART(COMPOSANT[1,1],COMPOSANT[3,1]);
                IF C <> ')' THEN ERREUR(11)

            END
```

```
    ELSE BEGIN

        NBCSP:=1;
        WHILE C <> ')' DO

        BEGIN

            IF C <> ',' THEN ERREUR(12);
            LIREC;
            LIREMOT;
            IF NBCSP = NBMAXCDES THEN ERREUR(309);
            IF RECHCOMP(MOT,NBSETP) <> 0 THEN ERREUR(102);
            NBCSP:=NBCSP+1;
            COMPOSANT[NBCSP]:=MOT

        END

    END;

    LIREC

END;

IF (C <> '&') AND (C <> '''') AND (C <> ':') AND (C <> '"') AND
   (C <> '!') AND (C <> '*')

THEN BEGIN

    IF C <> ';' THEN ERREUR(14);
    TYPSP:=INCONNU

     END

ELSE BEGIN

    CASE C OF

    '&' : TYPSP:=BSYMCIRC;
    ':' : TYPSP:=BSYM;
    '''': TYPSP:=BCIRC;
    '"' : BEGIN
              TYPSP:=OIDENT;
              IF NOT DEFCOMP AND (NBCSP > 1) THEN ERREUR(135)
          END;
    '!' : TYPSP:=BSSORD;
    '*' : TYPSP:=OREP

    END;

    LIREC;
    IF C <> ';' THEN ERREUR(15)

     END;

IF NOT EOLN(F) THEN ERREUR(6);
NUMLIGNE:=NUMLIGNE+1;
LIREC;
NUMCAR:=0;
LIREC;
IF C = '%' THEN

BEGIN

    LIREC;
    LIREMOT;
    DDEFCOMP:=FALSE;
    LIREC;
    IF C = '#' THEN
```

```
        BEGIN

            DDEFCOMP:=TRUE;
            LIREC;
            IF C <> ' ' THEN ERREUR(1);
            LIREC

        END

                ELSE

        BEGIN

            IF C <> '=' THEN ERREUR(2);
            LIREC;
            IF C <> ' ' THEN ERREUR(1);
            LIREC;
            IF C <> '(' THEN DECLSEC:=TRUE

        END

    END

            ELSE FINDECL:=TRUE

END

END;

(*************************************************************************
 *                                                                       *
 *   DEBUT DE LA PROCEDURE DE TRADUCTION                                 *
 *                                                                       *
 *      DES DECLARATIONS PRIMAIRES                                       *
 *                                                                       *
 *************************************************************************)

BEGIN

    DDEFCOMP:=FALSE;
    DECLSEC:=FALSE;
    LIREC;
    IF C <> '%' THEN ERREUR(5);
    LIREC;
    LIREMOT;
    IF C <> ' ' THEN ERREUR(1);
    LIREC;
    IF C = '#' THEN DDEFCOMP:=TRUE
               ELSE IF C <> '=' THEN ERREUR(2);
    LIREC;
    IF C <> ' ' THEN ERREUR(1);
    LIREC;
    REPEAT TRADLDP UNTIL FINDECL OR DECLSEC

END;

PROCEDURE TRADDS;

PROCEDURE TRADLDS;

(*************************************************************************
 *                                                                       *
 * DEBUT DE LA TRADUCTION D'UNE LIGNE                                    *
 *                                                                       *
 *   DE DECLARATION SECONDAIRE                                           *
 *                                                                       *
 *************************************************************************)
```

```
AR FINI : BOOLEAN;


EGIN

   IF RANG(MOT) <> 0 THEN ERREUR(101);
   IF NBSETS = NBMAXSP THEN ERREUR(303);
   NBSETS:=NBSETS+1;

   WITH SETS[NBSETS] DO

   BEGIN

        NOMSS:=MOT;
        NBSCSS:=0;
        COMPOIDENT:=FALSE;
        FINI:=FALSE;

        REPEAT BEGIN

             IF C <> '%' THEN ERREUR(5);
             IF NBSCSS = NBMAXSCSS THEN ERREUR(305);
             NBSCSS:=NBSCSS+1;
             LIREC;
             LIREMOT;
             SETC[NBSCSS]:=RANG(MOT);
             IF SETC[NBSCSS] = 0 THEN ERREUR(111);

             IF SETC[NBSCSS] <= 100

             THEN BEGIN

                  IF NOT SETP[SETC[NBSCSS]].DEFCOMP THEN ERREUR(212);
                  IF SETP[SETC[NBSCSS]].TYPSP = INCONNU
                      THEN SETP[SETC[NBSCSS]].TYPSP:=OSSREP;
                  IF (SETP[SETC[NBSCSS]].TYPSP <> OSSREP) AND
                     (SETP[SETC[NBSCSS]].TYPSP <> OIDENT)        THEN ERREUR(211);
                  IF SETP[SETC[NBSCSS]].TYPSP = OIDENT THEN COMPOIDENT:=TRUE

             END

             ELSE IF SETS[NBSCSS-100].COMPOIDENT THEN COMPOIDENT:=TRUE;

             IF C = ';' THEN FINI:=TRUE

                        ELSE

             BEGIN

                  IF C <> ' ' THEN ERREUR(1);
                  LIREC;
                  IF C <> '+' THEN ERREUR(10);
                  LIREC;
                  IF C <> ' ' THEN ERREUR(1);
                  LIREC

             END

        END UNTIL FINI;

        IF NOT EOLN(F) THEN ERREUR(6);
        NUMLIGNE:=NUMLIGNE+1;
        LIREC;
        NUMCAR:=0;
        LIREC;
        IF C = '%' THEN
```

```
      BEGIN

          LIREC;
          LIREMOT;
          IF C <> ' ' THEN ERREUR(1);
          LIREC;
          IF C <> '=' THEN ERREUR(8);
          LIREC;
          IF C <> ' ' THEN ERREUR(1);
          LIREC

      END

                ELSE FINDECL:=TRUE

   END

ND;

*************************************************************************
*                                                                       *
* DEBUT DE LA TRADUCTION DES                                            *
*                                                                       *
*  DECLARATIONS SECONDAIRES                                             *
*                                                                       *
*************************************************************************)

EGIN

   REPEAT TRADLDS UNTIL FINDECL

ND;

*************************************************************************
*                                                                       *
* DEBUT DE LA TRADUCTION DES DECLARATIONS                               *
*                                                                       *
*************************************************************************)


EGIN

   FINDECL:=FALSE;
   NBSETP:=0;NBSETS:=0;NBSUBSUT:=0;
   TRADDP;
   IF DECLSEC THEN TRADDS

ND;
```

```
PROCEDURE TRADACT;

VAR FINTEXTE : BOOLEAN;

(*-------------------------------------------------------------------------*)

FUNCTION CREATEUR(SSENS : INTEGER;OBJETS : BOOLEAN):INTEGER;

VAR I : INTEGER;

    TROUVE : BOOLEAN;

BEGIN

    I:=1;
    TROUVE:=FALSE;

    WHILE (I <= NBACTION) AND NOT TROUVE DO

        IF ACTION[I].ENS[OBJETS].NUMSSD = SSENS

            THEN TROUVE:=TRUE
            ELSE I:=I+1;

    IF NOT TROUVE THEN ERREUR(133);

    CREATEUR:=I

END;

(*-------------------------------------------------------------------------*)

PROCEDURE TRADLACT;

PROCEDURE EXPRESSION(OBJETS : BOOLEAN);

VAR FINLIGNE : BOOLEAN;

    SETOBJETS : INTEGER;

PROCEDURE DESIGNATION(OBJETS : BOOLEAN);

(*************************************************************************
 *                                                                       *
 * PROCEDURE DE DESIGNATION D'OBJETS                                     *
 *                                                                       *
 *          OU DE BOITES                                                 *
 *                                                                       *
 *************************************************************************)

BEGIN

    LIREC;
    LIREMOT;
    IF NBSUBSUT = NBMAXSUBSUT THEN ERREUR(304);
    IF RANG(MOT) > 200 THEN ERREUR(103);
    NBSUBSUT:=NBSUBSUT+1;
    ACTION[NBACTION].ENS[OBJETS].NUMSSD:=NBSUBSUT;

    WITH SUBSUT[NBSUBSUT] DO

    BEGIN

        COBJETS:=OBJETS;
        NOMSUBSUT:=MOT

    END;
```

```
    IF C <> ' ' THEN ERREUR(1);
    LIREC;
    IF C <> '=' THEN ERREUR(8);
    LIREC;
    IF C <> ' ' THEN ERREUR(1);
    LIREC

END;

FUNCTION NBPOS : INTEGER;

VAR VINBPOS : INTEGER;

(*-------------------------------------------------------------------------*)

FUNCTION NBELTENS(NUMSET : INTEGER) : INTEGER;

VAR I,VINBELTENS : INTEGER;

BEGIN

    IF NUMSET <= 100 THEN VINBELTENS:=SETP[NUMSET].NBCSP

                     ELSE WITH SETS[NUMSET-100] DO BEGIN

        VINBELTENS:=0;
        FOR I:=1 TO NBSCSS DO VINBELTENS:=VINBELTENS + NBELTENS(SETC[I])

                     END;

    NBELTENS:=VINBELTENS

END;

(*-------------------------------------------------------------------------*)

FUNCTION NBELTSENS(APPART : BOOLEAN) : INTEGER;

VAR I,VARINT : INTEGER;

BEGIN

    WITH ACTION[NBACTION].ENS[OBJETS].SOUSENS[APPART] DO

    CASE T OF

        1 : NBELTSENS:=NBCSUITE;

        2 : NBELTSENS:=NBCSORD;

        3 : IF DEFDEP

            THEN NBELTSENS:=ACTION[CREATEUR(SSBASE,OBJETS)].ENS[OBJETS].NBCCH

            ELSE BEGIN

                VARINT:=0;
                FOR I:=1 TO NBSSLIMACT DO
                VARINT:=VARINT +
                        ACTION[CREATEUR(ENSSS[I],OBJETS)].ENS[OBJETS].NBCCH;
                NBELTSENS:=VARINT

            END

    END

END;
```

```
(*************************************************************************
*                                                                        *
* PROCEDURE DE CALCUL DU NOMBRE                                          *
*                                                                        *
* D'ELEMENTS SUSCEPTIBLES D'ETRE                                         *
*                                                                        *
*           CHOISIS                                                      *
*                                                                        *
*************************************************************************)

BEGIN

   CASE ACTION[NBACTION].ENS[OBJETS].RESTR OF

      ABSENTE     : NBPOS:=NBELTENS(ACTION[NBACTION].ENS[OBJETS].SSET);

      APP         : NBPOS:=NBELTSENS(TRUE);

      NONAPP      : NBPOS:=NBELTENS(ACTION[NBACTION].ENS[OBJETS].SSET)
                           - NBELTSENS(FALSE);

      APPETNONAPP : BEGIN

          VINBPOS:=NBELTSENS(TRUE) - NBELTSENS(FALSE);
          IF VINBPOS < 1 THEN ERREUR(123);
          NBPOS:=VINBPOS

      END

   END

END;

PROCEDURE RESTRAPP(APPART : BOOLEAN);


PROCEDURE SUITEORDONNEE;

(*************************************************************************
 *                                                                       *
 * PROCEDURE DE SUITEORDONNEE                                            *
 *                                                                       *
 *************************************************************************)

BEGIN

   WITH ACTION[NBACTION].ENS[OBJETS] DO

   BEGIN

      IF SSET > 100 THEN ERREUR(213);

      WITH SOUSENS[APPART] DO

      BEGIN

          T:=2;
          LIREC;
          LIRENOMBRE(TYPEN);
          IF TYPEN = NUM THEN ELTSORD[1]:=NOMBRE
                         ELSE ERREUR(28);
          IF ELTSORD[1] > SETP[ACTION[NBACTION].ENS[OBJETS].SSET].NBCSP
              THEN ERREUR(114);

          IF C = '.' THEN
```

```
            BEGIN

                LIREC;
                IF C <> '.' THEN ERREUR(9);
                ELTSORD[2]:=0;
                LIREC;
                LIRENOMBRE(TYPEN);
                IF TYPEN = NUM THEN ELTSORD[3]:=NOMBRE
                               ELSE ERREUR(28);
                IF ELTSORD[3] > SETP[ACTION[NBACTION].ENS[OBJETS].SSET].NBCSP
                    THEN ERREUR(114);
                NBCSORD:=ELTSORD[3]-ELTSORD[1]+1;
                IF NBCSORD < 1 THEN ERREUR(124);
                IF C <> ']' THEN ERREUR(16)

            END

                      ELSE

            BEGIN

                NBCSORD:=1;
                WHILE C <> ']' DO

                BEGIN

                    IF C <> ',' THEN ERREUR(17);
                    IF NBCSORD = NBMAXCSORD THEN ERREUR(310);
                    NBCSORD:=NBCSORD+1;
                    LIREC;
                    LIRENOMBRE(TYPEN);
                    IF TYPEN = NUM THEN ELTSORD[NBCSORD]:=NOMBRE
                                   ELSE ERREUR(28);
                    IF ELTSORD[3] > SETP[ACTION[NBACTION].ENS[OBJETS].SSET].NBCS
                        THEN ERREUR(114);

                END
            END
        END
    END
END;

PROCEDURE SUITEDESIGNEE;

{*************************************************************************
 *                                                                       *
 * PROCEDURE DE SUITE DESIGNEE                                           *
 *                                                                       *
 *************************************************************************}

VAR NOMSUBSINT : INTEGER;

FUNCTION RECHNUMSUB : INTEGER;

VAR X : INTEGER;

BEGIN

    IF C <> '%' THEN ERREUR(5);
    LIREC;
    LIREMOT;
    X:=RANG(MOT);
    IF X <= 200 THEN ERREUR(112);
    X:=X-200;

    RECHNUMSUB:=X

END;
```

```
EGIN
   WITH ACTION[NBACTION].ENS[OBJETS].SOUSENS[APPART] DO

   BEGIN

      T:=3;
      NOMSUBSINT:=RECHNUMSUB;
      IF ACTION[CREATEUR(NOMSUBSINT,OBJETS)].ENS[OBJETS].SSET <>
         ACTION[NBACTION].ENS[OBJETS].SSET THEN ERREUR(134);

      IF C = ')'

      THEN BEGIN

          DEFDEP:=FALSE;
          ENSSS[1]:=NOMSUBSINT;
          NBSSLIMACT:=1

      END

      ELSE BEGIN

          IF C <> ' ' THEN ERREUR(18);
          LIREC;

          IF C = '-'

          THEN BEGIN

          DEFDEP:=TRUE;
          SSBASE:=NOMSUBSINT;
          PLUS:=FALSE;
          LIREC;
          IF C <> ' ' THEN ERREUR(1);
          LIREC;
          LIRENOMBRE(TYPEN);
          IF TYPEN = NUM THEN DEP:=NOMBRE
                         ELSE ERREUR(28);
          IF C <> ')' THEN ERREUR(11)

          END

          ELSE BEGIN

              IF C <> '+' THEN ERREUR(19);
              LIREC;
              IF C <> ' ' THEN ERREUR(1);
              LIREC;

              IF C <> '%'

              THEN BEGIN

                  DEFDEP:=TRUE;
                  SSBASE:=NOMSUBSINT;
                  PLUS:=TRUE;
                  LIRENOMBRE(TYPEN);
                  IF TYPEN = NUM THEN DEP:=NOMBRE
                                 ELSE ERREUR(28);
                  IF C <> ')' THEN ERREUR(11)

              END
```

```
            ELSE BEGIN

                ENSSS[1]:=NOMSUBSINT;
                ENSSS[2]:=RECHNUMSUB;
                NBSSLIMACT:=2;

                WHILE C <> ')' DO

                BEGIN

                    IF C <> ' ' THEN ERREUR(1);
                    LIREC;
                    IF C <> '+' THEN ERREUR(10);
                    LIREC;
                    IF C <> ' ' THEN ERREUR(1);
                    LIREC;
                    IF NBSSLIMACT = NBMAXSSLIMACT THEN ERREUR(792);
                    NBSSLIMACT:=NBSSLIMACT+1;
                    ENSSS[NBSSLIMACT]:=RECHNUMSUB

                END

            END

        END

    END

END;

PROCEDURE SUITE;

{*************************************************************************
 *                                                                       *
 * PROCEDURE DE SUITE                                                    *
 *                                                                       *
 *************************************************************************}

VAR X : INTEGER;

BEGIN

    WITH ACTION[NBACTION].ENS[OBJETS].SOUSENS[APPART] DO

    BEGIN

        T:=1;
        X:=ACTION[NBACTION].ENS[OBJETS].SSET;
        IF X > 100 THEN ERREUR(214);
        IF SETP[X].DEFCOMP THEN ERREUR(136);

        IF SETP[X].COMPOSANT[2] = '.'

        THEN BEGIN

            LIREMOT;
            ELTSUITE[1]:=RECHINT(MOT,X);

            IF C = '.'
```

```
    THEN BEGIN

        LIREC;
        IF C <> '.' THEN ERREUR(9);
        ELTSUITE[2]:=0;
        LIREC;
        LIREMOT;
        ELTSUITE[3]:=RECHINT(MOT,X);
        NBCSUITE:=ELTSUITE[3]-ELTSUITE[1]+1;
        IF NBCSUITE < 2 THEN ERREUR(124);
        IF C <> ')' THEN ERREUR(11)

    END

    ELSE BEGIN

        NBCSUITE:=1;
        WHILE C <> ')' DO

        BEGIN

            IF C <> ',' THEN ERREUR(12);
            LIREC;
            LIREMOT;
            IF NBCSUITE = NBMAXCSUITE THEN ERREUR(311);
            NBCSUITE:=NBCSUITE+1;
            ELTSUITE[NBCSUITE]:=RECHINT(MOT,X)

        END

    END

END

ELSE BEGIN

    LIREMOT;
    ELTSUITE[1]:=RECHCOMP(MOT,X);
    IF ELTSUITE[1] = 0 THEN ERREUR(113);

    IF C = '.'

    THEN BEGIN

        LIREC;
        IF C <> '.' THEN ERREUR(9);
        ELTSUITE[2]:=0;
        LIREC;
        LIREMOT;
        ELTSUITE[3]:=RECHCOMP(MOT,X);
        IF ELTSUITE[3] = 0 THEN ERREUR(113);
        NBCSUITE:=ELTSUITE[3]-ELTSUITE[1]+1;
        IF NBCSUITE < 2 THEN ERREUR(124);
        IF C <> ')' THEN ERREUR(11)

    END

    ELSE BEGIN

        NBCSUITE:=1;

        WHILE C <> ')' DO
```

```
            BEGIN

                IF C <> ',' THEN ERREUR(12);
                LIREC;
                LIREMOT;
                IF NBCSUITE = NBMAXCSUITE THEN ERREUR(311);
                NBCSUITE:=NBCSUITE+1;
                ELTSUITE[NBCSUITE]:=RECHCOMP(MOT,X);
                IF ELTSUITE[NBCSUITE] = 0 THEN ERREUR(113)

            END

        END

    END

ND;

*************************************************************************
*                                                                       *
* PROCEDURE DE RESTRAPP                                                 *
*                                                                       *
*************************************************************************)

EGIN

    WITH ACTION[NBACTION].ENS[OBJETS] DO

    BEGIN

        IF APPART THEN RESTR:=APP

                  ELSE

        BEGIN

            IF RESTR = APP THEN RESTR:=APPETNONAPP
                           ELSE RESTR:=NONAPP

        END;

        LIREC;
        IF C <> ' ' THEN ERREUR(1);
        LIREC;

        IF C = '[' THEN SUITEORDONNEE

                   ELSE BEGIN

                            IF C <> '(' THEN ERREUR(20);
                            LIREC;

                            IF C = '%' THEN SUITEDESIGNEE

                                       ELSE SUITE

                        END;

        LIREC

    END

ND;
```

```
PROCEDURE PARTITION;

(*************************************************************************)
(*                                                                       *)
(* PROCEDURE DE PARTITION                                                *)
(*                                                                       *)
(*************************************************************************)

VAR I,TOTOBJ,TOTB,SETOBJ : INTEGER;

BEGIN

    LIREC;

    IF (SETP[ACTION[NBACTION].ENS[FALSE].SSET].TYPSP <> BSSORD) AND
       (SETP[ACTION[NBACTION].ENS[FALSE].SSET].TYPSP <> BORD) THEN ERREUR(221);

    SETOBJ:=ACTION[NBACTION].ENS[TRUE].SSET;

    IF SETOBJ > 100

        THEN IF SETS[SETOBJ-100].COMPOIDENT THEN ERREUR(222)

        ELSE IF SETP[SETOBJ].TYPSP <> OSSREP THEN ERREUR(222);


    WITH ACTION[NBACTION] DO BEGIN

        IF NOT TYPACT THEN ERREUR(223);
        IF SETOBJPREC THEN ERREUR(224);
        PART:=TRUE;

        WITH DESPART DO BEGIN

            NBPART:=0;
            TOTOBJ:=0;
            TOTB:=0;

            REPEAT BEGIN

                IF C <> ' ' THEN ERREUR(1);
                IF NBPART = NBMAXPARDES THEN ERREUR(306);
                NBPART:=NBPART+1;

                WITH TPART[NBPART] DO BEGIN

                    LIREC;
                    LIRENOMBRE(TYPEN);
                    IF TYPEN = NUM THEN NBFMEMBO:=NOMBRE
                                   ELSE ERREUR(28);
                    TOTB:=TOTB+NOMBRE;
                    IF C <> ':' THEN ERREUR(21);
                    LIREC;
                    LIRENOMBRE(TYPEN);
                    FOR I:=1 TO (NBPART-1) DO
                        IF TPART[I].NBODSB = NOMBRE THEN ERREUR(127);
                    IF TYPEN = NUM THEN NBODSB:=NOMBRE
                                   ELSE ERREUR(28);
                    TOTOBJ:=TOTOBJ + NBFMEMBO*NOMBRE

                END

            END UNTIL (C = ';') OR (C = '.');

        END

    END;
```

```
   IF TOTOBJ <> ACTION[NBACTION].ENS[TRUE].NBCCH THEN ERREUR(128);

   IF TOTB <> ACTION[NBACTION].ENS[FALSE].NBCCH THEN ERREUR(129)

ND;

ROCEDURE PRECISIONOBJETS;

*************************************************************************
*                                                                       *
* PROCEDURE DE PRECISION                                                *
*                                                                       *
*************************************************************************)

AR FINI, PASDEPLUS, TROUVE : BOOLEAN;

   TOTNBOPR,SETSL,I : INTEGER;

EGIN

   LIREMOT;
   IF MOT <> 'AVEC' THEN ERREUR(22);
   IF C <> ' ' THEN ERREUR(1);
   IF NOT ACTION[NBACTION].TYPACT THEN ERREUR(231);
   SETSL:=ACTION[NBACTION].ENS[TRUE].SSET;
   IF SETSL <= 100 THEN ERREUR(137)
                   ELSE IF SETS[SETSL-100].COMPOIDENT THEN ERREUR(232);
   ACTION[NBACTION].SETOBJPREC:=TRUE;

   WITH ACTION[NBACTION].DESSETPREC DO

   BEGIN

       NBSPSOBJ:=0;
       FINI:=FALSE;
       PASDEPLUS:=TRUE;
       TOTNBOPR:=0;

       REPEAT BEGIN

           IF NBSPSOBJ = NBMAXSPSOBJ THEN ERREUR(308);
           NBSPSOBJ:=NBSPSOBJ+1;
           LIREC;
           LIRENOMBRE(TYPEN);
           IF C <> ' ' THEN ERREUR(1);

           WITH DSETPREC[NBSPSOBJ] DO

           BEGIN

               IF TYPEN = NUM THEN NBOPR:=NOMBRE
                              ELSE ERREUR(28);
               TOTNBOPR:=TOTNBOPR+NBOPR;
               LIREC;
               SIGNOPR:=EXISTPAS;
               IF (C = '-') OR (C = '+') THEN

               BEGIN

                   IF C = '-' THEN SIGNOPR:=MOINS
                              ELSE BEGIN

                                       SIGNOPR:=PLUS;
                                       PASDEPLUS:=FALSE

                                   END;
```

```
                LIREC;
                IF C <> ' ' THEN ERREUR(1);
                LIREC

            END;

            LIREMOT;
            IF MOT <> 'DE' THEN ERREUR(23);
            IF C <> ' ' THEN ERREUR(1);
            LIREC;
            IF C <> '%' THEN ERREUR(5);
            LIREC;
            LIREMOT;
            IF C <> ' ' THEN ERREUR(1);
            SSSET:=RANG(MOT);
            IF (SSSET = 0) OR (SSSET > 200) THEN ERREUR(111);

            IF SSSET <= 100

            THEN BEGIN

                IF SETP[SSSET].TYPSP <> OSSREP THEN ERREUR(233);
                IF NBOPR > SETP[SSSET].NBCSP THEN ERREUR(126)

            END

            ELSE IF SETS[SSSET-100].COMPOIDENT THEN ERREUR(233);

            TROUVE:=FALSE;
            I:=1;

            REPEAT IF SSSET = SETS[SETSL-100].SETC[I]

                                THEN TROUVE:=TRUE
                                ELSE I:=I+1;

            UNTIL (I > SETS[SETSL-100].NBSCSS) OR TROUVE;

            IF NOT TROUVE THEN ERREUR(131)

        END;

        LIREC;
        IF C = 'E' THEN
        BEGIN

            LIREMOT;
            IF MOT <> 'ET' THEN ERREUR(24);
            IF C <> ' ' THEN ERREUR(1)

        END
                    ELSE FINI:=TRUE

    END UNTIL FINI;

    IF NOT PASDEPLUS THEN FOR I:=1 TO NBSPSOBJ
        DO IF DSETPREC[I].SIGNOPR = MOINS THEN ERREUR(236)


END;

IF PASDEPLUS THEN IF TOTNBOPR > ACTION[NBACTION].ENS[TRUE].NBCCH
                                                    THEN ERREUR(125)
END;
```

```
(*************************************************************************)
*                                                                         *
* PROCEDURE D'EXPRESSION D'OBJETS                                         *
*                                                                         *
*   ET DE BOITES                                                          *
*                                                                         *
(*************************************************************************)

BEGIN

   WITH ACTION[NBACTION].ENS[OBJETS] DO

   BEGIN

      NUMSSD:=0;
      IF C = '%' THEN DESIGNATION(OBJETS);
      LIRENOMBRE(TYPEN);
      IF TYPEN = NUM THEN NBCCH:=NOMBRE
                     ELSE NBCCH:=0;
      IF C <> ' ' THEN ERREUR(1);
      LIREC;
      LIREMOT;
      IF MOT <> 'DE' THEN ERREUR(23);
      IF C <> ' ' THEN ERREUR(1);
      LIREC;
      IF C <> '%' THEN ERREUR(5);
      LIREC;
      LIREMOT;
      SSET:=RANG(MOT);
      IF SSET = 0 THEN ERREUR(111);
      RESTR:=ABSENTE;

      IF OBJETS THEN

      BEGIN

          IF SSET <= 100

          THEN BEGIN

              IF SETP[SSET].TYPSP = INCONNU THEN SETP[SSET].TYPSP := OSSREP;

              IF (SETP[SSET].TYPSP <> OSSREP) AND
                 (SETP[SSET].TYPSP <> OIDENT) AND
                 (SETP[SSET].TYPSP <> OREP)        THEN ERREUR(132);

              IF NOT ACTION[NBACTION].TYPACT
                  THEN IF SETP[SSET].TYPSP <> OSSREP THEN ERREUR(241)

          END

          ELSE IF NOT ACTION[NBACTION].TYPACT
                    THEN IF SETS[SSET-100].COMPOIDENT THEN ERREUR(241);

          IF C <> ' ' THEN ERREUR(1);
          LIREC;
          ACTION[NBACTION].SETOBJPREC:=FALSE;

          IF C = 'A' THEN PRECISIONOBJETS;

          IF C = 'R' THEN BEGIN

                              RESTRAPP(TRUE);
                              IF C <> ' ' THEN ERREUR(1);
                              LIREC

                          END;
```

```
    IF C = 'M' THEN BEGIN

                        RESTRAPP(FALSE);
                        IF C <> ' ' THEN ERREUR(1);
                        LIREC

                    END;

END

              ELSE

BEGIN

    IF SSET <= 100

    THEN BEGIN

        IF SETP[SSET].TYPSP = INCONNU THEN SETP[SSET].TYPSP := BORD;

        IF (SETP[SSET].TYPSP <> BORD) AND
           (SETP[SSET].TYPSP <> BSSORD) AND
           (SETP[SSET].TYPSP <> BCIRC) AND
           (SETP[SSET].TYPSP <> BSYM) AND
           (SETP[SSET].TYPSP <> BSYMCIRC)       THEN ERREUR(132)

    END

    ELSE ERREUR(132);

    IF NOT ACTION[NBACTION].TYPACT THEN IF SETP[SSET].TYPSP <> BORD
        THEN ERREUR(235);

    IF ACTION[NBACTION].SETOBJPREC THEN IF SETP[SSET].TYPSP <> BSSORD
        THEN ERREUR(234);

    SETOBJETS:=ACTION[NBACTION].ENS[TRUE].SSET;

    IF SETOBJETS <= 100

    THEN BEGIN

        IF SETP[SETOBJETS].TYPSP = OREP

            THEN IF ((SETP[SSET].TYPSP <> BORD) AND
                     (SETP[SSET].TYPSP <> BSSORD))
                     THEN ERREUR(201)

            ELSE IF SETP[SETOBJETS].TYPSP = OIDENT

                THEN IF SETP[SSET].TYPSP <> BORD THEN ERREUR(202)

    END

    ELSE IF SETS[SETOBJETS-100].COMPOIDENT
              THEN IF SETP[SSET].TYPSP <> BORD THEN ERREUR(202);

    ACTION[NBACTION].PART:=FALSE;

    FINLIGNE:=FALSE;

    IF C <> ' ' THEN FINLIGNE:=TRUE

                ELSE LIREC;
```

```
        IF NOT FINLIGNE THEN IF C = 'R' THEN

        BEGIN

            RESTRAPP(TRUE);
            IF C <> ' ' THEN FINLIGNE:=TRUE
                        ELSE LIREC

        END;

        IF NOT FINLIGNE THEN IF C = 'M' THEN

        BEGIN

            RESTRAPP(FALSE);
            IF C <> ' ' THEN FINLIGNE:=TRUE
                        ELSE LIREC

        END;

        IF NOT FINLIGNE THEN IF C = ',' THEN PARTITION

    END;

    IF NOT OBJETS THEN IF (ACTION[NBACTION].TYPACT) AND
                          (NOT ACTION[NBACTION].PART)
        THEN IF ACTION[NBACTION].ENS[TRUE].NBCCH <>
                ACTION[NBACTION].ENS[FALSE].NBCCH   THEN ERREUR(121);

    NBCP:=NBPOS;
    IF NBCCH = 0 THEN NBCCH:=NBCP;

    IF OBJETS

    THEN BEGIN

        IF SSET <= 100

        THEN BEGIN

            IF SETP[SSET].TYPSP <> OREP
                THEN IF NBCP < NBCCH THEN ERREUR(122)

        END

        ELSE IF NBCP < NBCCH THEN ERREUR(122)

    END

    ELSE IF NBCP < NBCCH THEN ERREUR(122)

  END

ND;

ROCEDURE CALCULERSOL;BEGIN END;

{*************************************************************************
*                                                                        *
* PROCEDURE DE TRADUCTION D'UNE                                          *
*                                                                        *
*     LIGNE D'ACTION                                                     *
*                                                                        *
*************************************************************************}
```

```
BEGIN
    NBACTION:=NBACTION+1;
    WITH ACTION[NBACTION] DO
    BEGIN
        COORD:=ET;
        LIREMOT;
        IF (MOT = 'OU') OR (MOT = 'SANS') THEN
        BEGIN
            IF MOT = 'OU' THEN COORD:=OU
                          ELSE COORD:=SANS;
            IF C <> ' ' THEN ERREUR(1);
            LIREC;
            LIREMOT
        END
        IF MOT = 'PLACER' THEN TYPACT:=TRUE
                          ELSE IF MOT = 'REPARTIR'
                                  THEN TYPACT:=FALSE
                                  ELSE ERREUR(25);
        IF C <> ' ' THEN ERREUR(1);
        LIREC;
        EXPRESSION(TRUE);
        LIREMOT;
        IF MOT <> 'DANS' THEN ERREUR(26);
        IF C <> ' ' THEN ERREUR(1);
        LIREC;
        EXPRESSION(FALSE);
        CALCULERSOL;
        IF C = '.' THEN FINTEXTE:=TRUE
                   ELSE
        BEGIN
            IF C <> ';' THEN ERREUR(27);
            IF NBACTION = NBMAXACTION THEN ERREUR(301);
            IF NOT EOLN(F) THEN ERREUR(6);
            NUMLIGNE:=NUMLIGNE+1;
            LIREC;
            NUMCAR:=0;
            LIREC
        END
    END
END;

(*************************************************************************
 *                                                                       *
 *   DEBUT DE LA PROCEDURE DE                                            *
 *                                                                       *
 *    TRADUCTION DES ACTIONS                                             *
 *                                                                       *
 *************************************************************************)

BEGIN
    FINTEXTE:=FALSE;
    NBACTION:=0;
    REPEAT TRADLACT UNTIL FINTEXTE;
END;
```

```
ROCEDURE TESTSET;

EGIN

   FOR I:=1 TO NBSETP DO

   WITH SETP[I] DO

   BEGIN

       WRITELN(NOMSP);
       WRITELN(NBCSP);
       CASE TYPSP OF

            INCONNU   : WRITELN('INCONNU');
            OSSREP    : WRITELN('OSSREP');
            OREP      : WRITELN('OREP');
            OIDENT    : WRITELN('OIDENT');
            BORD      : WRITELN('BORD');
            BSSORD    : WRITELN('BSSORD');
            BCIRC     : WRITELN('BCIRC');
            BSYM      : WRITELN('BSYM');
            BSYMCIRC  : WRITELN('BSYMCIRC')

       END;
       IF DEFCOMP THEN WRITELN('VRAI')
                  ELSE WRITELN('FAUX');
       IF NOT DEFCOMP THEN

       BEGIN

            WRITELN(COMPOSANT[1]);
            IF COMPOSANT[2] = '.' THEN

            BEGIN

                 WRITELN('INTERVALLE');
                 WRITELN(COMPOSANT[3])

            END

                                  ELSE

            FOR J:=2 TO NBCSP DO WRITELN(COMPOSANT[J]);

       END

   END;

   FOR I:=1 TO NBSETS DO

   WITH SETS[I] DO

   BEGIN

       WRITELN(NOMSS);
       IF COMPOIDENT THEN WRITELN('OIDENT');
       WRITELN(NBSCSS);
       FOR J:=1 TO NBSCSS DO WRITELN(SETC[J])

   END;
```

```
    FOR I:=1 TO NBSUBSUT DO

    WITH SUBSUT[I] DO

    BEGIN

        WRITELN(NOMSUBSUT);
        IF COBJETS THEN WRITELN('OBJETS')
                   ELSE WRITELN('BOITES')

    END;

END;
(*-------------------------------------------------------------------------------*)
PROCEDURE TESTACTION;

PROCEDURE TESTSOUSENS(BB:BOOLEAN);

BEGIN

    WITH ACTION[I].ENS[B].SOUSENS[BB] DO

    BEGIN

        CASE T OF

            1 : BEGIN

                WRITELN('SUITE');
                WRITELN(NBCSUITE);
                FOR J:=1 TO NBCSUITE DO WRITELN(ELTSUITE[JJ])

            END;

            2 : BEGIN

                WRITELN('SUITEORD');
                WRITELN(NBCSORD);
                FOR J:=1 TO NBCSORD DO WRITELN(ELTSORD[J])

            END;

            3 : BEGIN

                WRITELN('SUITEDES');

                IF DEFDEP THEN BEGIN

                    WRITELN('DEPLACEMENT');
                    WRITELN(SSBASE);
                    IF PLUS THEN WRITELN('PLUS');
                    WRITELN(DEP)

                END

                ELSE BEGIN

                    WRITELN(NBSSLIMACT);
                    FOR J:=1 TO NBSSLIMACT DO WRITELN(ENSSS[J])

                END
            END
        END
    END
END;
```

```
*-------------------------------------------------------------------------------*)

EGIN

   FOR I:=1 TO NBACTION DO

   WITH ACTION[I] DO

   BEGIN

       CASE COORD OF

            ET   : WRITELN('ET');
            OU   : WRITELN('OU');
            SANS : WRITELN('SANS')

       END;

       IF TYPACT THEN WRITELN('PLACER')
                 ELSE WRITELN('REPARTIR');

       FOR B:=FALSE TO TRUE DO

       WITH ENS[B] DO

       BEGIN

           WRITELN(SSET);
           WRITELN(NBCCH);
           WRITELN(NBCP);
           WRITELN(NUMSSD);

           CASE RESTR OF

               APP : BEGIN

                   WRITELN('APP');

                   TESTSOUSENS(TRUE)

               END;

               NONAPP : BEGIN

                   WRITELN('NONAPP');

                   TESTSOUSENS(FALSE)

               END;

               APPETNONAPP : BEGIN

                   WRITELN('APPETNONAPP');

                   TESTSOUSENS(TRUE);

                   TESTSOUSENS(FALSE)

               END

           END

       END;
```

```
    IF PART THEN WITH DESPART DO

    BEGIN

        WRITELN('PARTITION');
        WRITELN(NBPART);

        FOR J:=1 TO NBPART DO WITH TPART[J] DO

        BEGIN

            WRITELN(NBODSB);
            WRITELN(NBFMEMBO)

        END

    END;

    IF SETOBJPREC THEN WITH DESSETPREC DO

    BEGIN

        WRITELN('PRECISION');
        WRITELN(NBSPSOBJ);

        FOR J:=1 TO NBSPSOBJ DO WITH DSETPREC[J] DO

        BEGIN

            WRITELN(SSSET);
            WRITELN(NBOPR);
            IF SIGNOPR = PLUS THEN WRITELN('PLUS');
            IF SIGNOPR = MOINS THEN WRITELN('MOINS');
            IF SIGNOPR = EXISTPAS THEN WRITELN('EXISTPAS')

        END

    END

END

ND;
```